HUMAN SOCIAL DYNAMICS MULTI AGENT SYSTEM

By

Oralee N. Nudson

RECOMMENDED:

_____

Dr. David Newman
Professor, Department of Physics, Space Physics and Aeronomy
University of Alaska Fairbanks

_____

Dr. Brian Hay
ASSERT Lab Director
Assistant Professor, Department of Computer Science
University of Alaska Fairbanks

_____

Dr. Kara Nance
Advisory Committee Chair
Chair and Professor, Department of Computer Science
University of Alaska Fairbanks

_____

Date

HUMAN SOCIAL DYNAMICS MULTI AGENT SYSTEM

A

PROJECT

Presented to the Faculty

of the University of Alaska Fairbanks

In Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

By

Oralee N. Nudson, B.S., B.S.

Fairbanks, Alaska

May 2009

Abstract

Current political and economic events are placing an emphasis on energy production and consumption more than ever before. This leads to the necessity for continued research with power distribution systems and factors influencing system operation. The Human Social Dynamics Multi Agent System (HSDMAS) is a project contributing to the study of power distribution networks. By examining power failures as a string of related events while incorporating intelligent learning agents representing human factors, the HSDMAS takes a unique approach towards the understanding and prevention of large scale power failures by coupling a probabilistic model of load-dependent cascading failure, CASCADE, with a dynamic power systems model, OPA. The HSDMAS project focuses on improving and optimizing the performance of the CASCADE and OPA models individually, then develops an interactive multi-layer, multi-agent system modeling power transmission and human factors represented by utility optimization.

Table of Contents

Page

Page

List of Figures

Page

List of Tables

Page

List of Appendices

Page

Acknowledgments

This work was partially funded through the National Science Foundation Grant "DHB Collaborative Research: Human Decision Making Dynamics and its Impact on Infrastructure Systems" (NSF-SES-062361).

This work was also supported in part by a grant of HPC resources from the Arctic Region Supercomputing Center (ARSC) at the University of Alaska Fairbanks as part of the Department of Defense High Performance Computing Modernization Program.

Special "Thank Yous" are directed towards my UAF Graduate Committee Members: Dr. David Newman, Dr. Kara Nance, and Dr. Brian Hay for answering my many questions and providing invaluable direction; Mr. Don Bahls with ARSC for sharing his superior knowledge in all areas of Computer Science; Ms. Barbara Horner-Miller and ARSC for allowing me to pursue my graduate studies in addition to working fulltime; the UAF Computer Science Department for educating and challenging those interested in learning; the University of Alaska Fairbanks for waiving the tuition of staff members; and to Mr. Jason Focke for his infinite and unconditional support.

Chapter 1 Introduction

1.1 Problem Statement

Large scale power outages continue to cause disasters resulting in loss of life, property, and productivity. Because of this, further research must continue to increase the understanding of power distribution systems and factors influencing their operation.

1.2 Project Description

The Human Social Dynamics Multi Agent System (HSDMAS) is a project contributing to the study of power distribution networks. By examining power failures as a string of related events while incorporating intelligent learning agents representing human factors, the HSDMAS takes a unique approach towards the understanding and prevention of large scale power failures.

1.3 Motivation

Today's society is built on an infrastructure unsustainable without power. Therefore, it is in our best interest to further the understanding of power transmission systems and human factors influencing them. Whether power generation and supply is intentionally manipulated for profit as demonstrated in the 2000 California energy crisis[7], or natural disasters causing extended blackouts result in deadly consequences like those of the 2008 Northeast winter storms[15], the evidence of the need to improve existing power distribution networks and influence the design of future networks is obvious. It is the intent of the HSDMAS to address these needs.

1.4 Goals

Developed for the "Human Decision Making Dynamics and its Impact on Infrastructure Systems" National Science Foundation research proposal[9], the HSDMAS couples a probabilistic model of load dependent cascading failure, CASCADE[2], with a dynamic power systems model, OPA[5]. The goals of the HSDMAS project are to improve and optimize the performance of the CASCADE probabilistic model, to improve and optimize the performance of the dynamic power systems model OPA, and to couple CASCADE and OPA together to develop an interactive multi-layer, multi-agent system modeling power transmission and human factors represented by utility optimization.

Chapter 2 Cascade Model

2.1 Program Description

The CASCADE model is a probabilistic model of load dependent cascading failure written in C++. Within this model, a network of interconnected independent components is created. Each component is assigned a random load representing the stress accumulated on that one component. Should the component fail due to old age or being overloaded, the failed component's load is transferred to a random subset of other components. The cascading event occurs when the transferred load causes the load recipient components to also overload and subsequently fail. This results in a chain reaction, or cascade, because the load recipient components then shed their loads to other operable components. Component replacement and upgrades allow the system to recover from and prevent future incapacitating cascades during each iteration of the model's cycle[3]. Figure 1 shows the flow of control for the CASCADE model. The CASCADE model source code is included in Appendix 1.

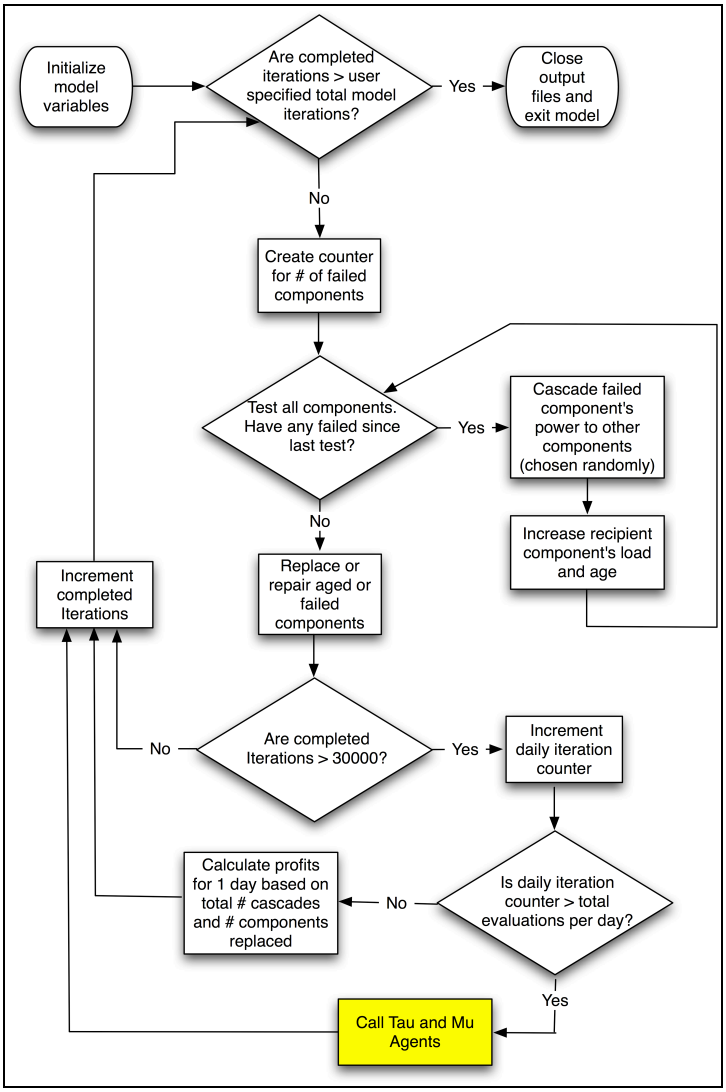

Figure 1: CASCADE Model Flow Chart

Two independent parameters, `mu` and `tau`, operate in the CASCADE model. The `mu` parameter represents the upgrade rate of the system components. The `tau` parameter represents the replacement time of the aged components. These two parameters are controlled by intelligent agent functions which interact with the system to learn to maximize their respective utility functions. A maximized utility function creates the highest overall profit by selling component services (minus the cost of component maintenance and upgrade expenses.) Each agent receives penalties for random and overloaded component failures, but can reduce the probability of

component failure by replacing components before the element of age or the probability for overloading becomes significant.

2.2 Benchmarking

Table 1 shows benchmarking results for the CASCADE model run with one million iterations using various compilers and computer architectures. The systems included a PowerPC Macintosh laptop, an Intel MacBook, and the Sun Opteron 2312-processor Cluster (Midnight). All three machines were available through the Arctic Region Supercomputing Center (ARSC). From the results obtained, it is obvious that the CASCADE model timing is greatly influenced by the particular architecture on which it is run.

Table 1: Benchmarking Results for CASCADE

Compiler	Optimization	Architecture	Time in Minutes
g++ 4.0.1	None	1.5 GHz Power PC G4	10.22
g++ 4.0.1	None	2.4 GHz Intel Dual Core, 4GB DDR2 SDRAM	2.63
g++ 4.0.1	-O3		2.19
g++ 3.3.3	None	2.6 GHz Sun Opteron Quad Core, 8 GB per socket DDR2-667 RAM	4.95
g++ 3.3.3	-O3		4.56
pathCC 3.1	None		2.88
pathCC 3.1	-O3		2.77
pgCC 7.2.2	None		2.61
pgCC 7.2.2	-O3		2.55

2.3 Modularization

The original CASCADE model consisted of a four hundred and forty-two line main function along with two other functions representing the `mu` and `tau` agents. The first step in improving the model source code was to simplify the very long main function by restructuring it to thirty-six lines and creating an additional six functions. This program modularization aided in program readability, isolated file input and output, and eased debugging efforts. Working with a more function oriented program also assisted in identifying the model's overall flow of control and eased the profiling and optimization effort.

2.4 Profiling

The CASCADE model was profiled using a Sun Opteron dual socket dual core processor and the GNU gprof software version 2.15.90.0.1.1. To use this profiling tool, the `-pg` flag was added to code compilation then the program was run as normal. Because of the additional `-pg` compiler flag, a "gmon.out" file was produced along with the normal CASCADE output files. To view the results of the gprof profiler, the `gprof cascadeModel gmon.out` command was executed in the directory containing the executable and "gmon.out" files.

The flat profile generated by gprof (as shown in Table 2) reveals that 99.75% of total CASCADE model execution time was spent in the *timeAdvance* function. Unfortunately the *timeAdvance* function was approximately two hundred and fifty lines long, therefore the TAU Portable Profiling Package[16] was used to narrow down the exact lines of code consuming the majority of CPU time.

Table 2: Flat Profile of CASCADE Generated by Gprof

% time	Cumulative seconds	Self seconds	Self calls	Total us/call	Us/call	Name
99.75	59.93	59.93	1000000	59.93	59.93	timeAdvance()
0.12	60.00	0.07	9699	7.22	7.22	Agent1()
0.10	60.06	0.06	9699	6.19	6.19	Agent2()
0.05	60.09	0.03	9699	3.09	3.09	Agent3()
0.02	60.10	0.01	960300	0.01	0.01	updateProfits()
0.00	60.10	0.00	897476	0.00	0.00	Stdio::precision(long)
0.00	60.10	0.00	9699	0.00	0.00	afterAgents()
0.00	60.10	0.00	9699	0.00	0.00	beforeAgents()
0.00	60.10	0.00	12	0.00	0.00	Std:: Ios Openmode
0.00	60.10	0.00	1	0.00	0.00	Global constructors
0.00	60.10	0.00	1	0.00	0.00	closeFiles()
0.00	60.10	0.00	1	0.00	0.00	InitializeSystem()
0.00	60.10	0.00	1	0.00	0.00	Static_init_destruct()

Use of the TAU profiler was very straight forward. After adding the `#include <TAU.h>` statement to the CASCADE source code, the `TAU_START("my label")` and `TAU_STOP("my label")` flags were placed around the three largest for-loops in the *timeAdvance* function. The program was then compiled using a generic TAU Makefile (located

in Appendix 2) then run normally. An output file named "profile.0.0.0" was generated and its contents were viewed with the `pprof profile.0.0.0` command. The TAU profiling results are shown in Table 3.

Table 3: Profiling Results from Tau Profiling Software

% Time	Exclusive msec	Inclusive total msec	# Call	# Subrs	Inclusive usec/call	Name
100.0	910	1:57.280	1	1.1485E+06	117280529	main()
98.8	12807	1:55.894	1E+06	1E+06	116	timeAdvance()
87.9	1:43.086	1:43.086	1E+06	0	103	"testing components for failure"
0.2	276	276	9699	0	28	afterAgents()
0.1	71	71	100001	0	1	updateProfits()
0.0	40	40	9699	0	4	Agent3()
0.0	34	34	9699	0	4	Agent1()
0.0	20	20	1	0	20920	closeFiles()
0.0	19	19	9699	0	2	Agent2()
0.0	8	8	9699	0	1	beforeAgents()
0.0	4	4	1	0	4686	initializeSystem()

Of the three for-loops analyzed with TAU ("testing components for failure", "replace repair", and "cascading event"), the "testing components for failure" section consumed the majority of CPU time. It can be seen from the "profile.0.0.0" output file that the "testing components for failure" function took 87% of the total CPU time. The contents of this particular for-loop are shown in Figure 2.

```
for(int i=0; i< Size ; i++)
{  double rf       = rand()/g;
   double xt       = Age[i];
   double xt2      = xt*xt;
   double test     = pow(FailR, xt2);
   Load[i]         = MeanLoad*(1+2.*(rand()/g-0.5)*LoadRange);
   TotalLoad       = TotalLoad + Load[i];
   if (rf > test || Load[i]>MaxLoad[i]) // Component i fails
   {
      Component[i]    = 0;
      WaitTime        = j-LastFail[i];
      LastFail[i]     = j;
      fout.precision(10);
      fout << i << "\t" << WaitTime << "\n";
      fout.clear();
      Tfails          += 1;
   }
   Time[i]  += 1;
   TimeM[i] += 1;
   Age[i]   += 1;
}
```

Figure 2: Contents of Function Consuming Most CASCADE CPU Time

Within this particular for-loop, two segments of code were identified to be the most likely candidates for consuming CPU time. These segments were the *pow* math library and the *fout* file output function calls. Determining which of these two functions consumed the most time required the use of another profiling tool; one which could identify CPU consumption line-by-line. The Instruments[6] software handled this task quite well, and successfully profiled the CASCADE model on a line-by-line basis. Figure 3 shows a screen shot of the Instruments GUI in production while profiling a typical CASCADE run. The results achieved from the Instruments output confirmed the previous TAU results in addition to showing that the *pow* math function consumed a significant portion of CPU time. Therefore, it was obvious that replacing the *pow* function call with a faster combination of mathematical operations was the starting point for CASCADE model optimization.

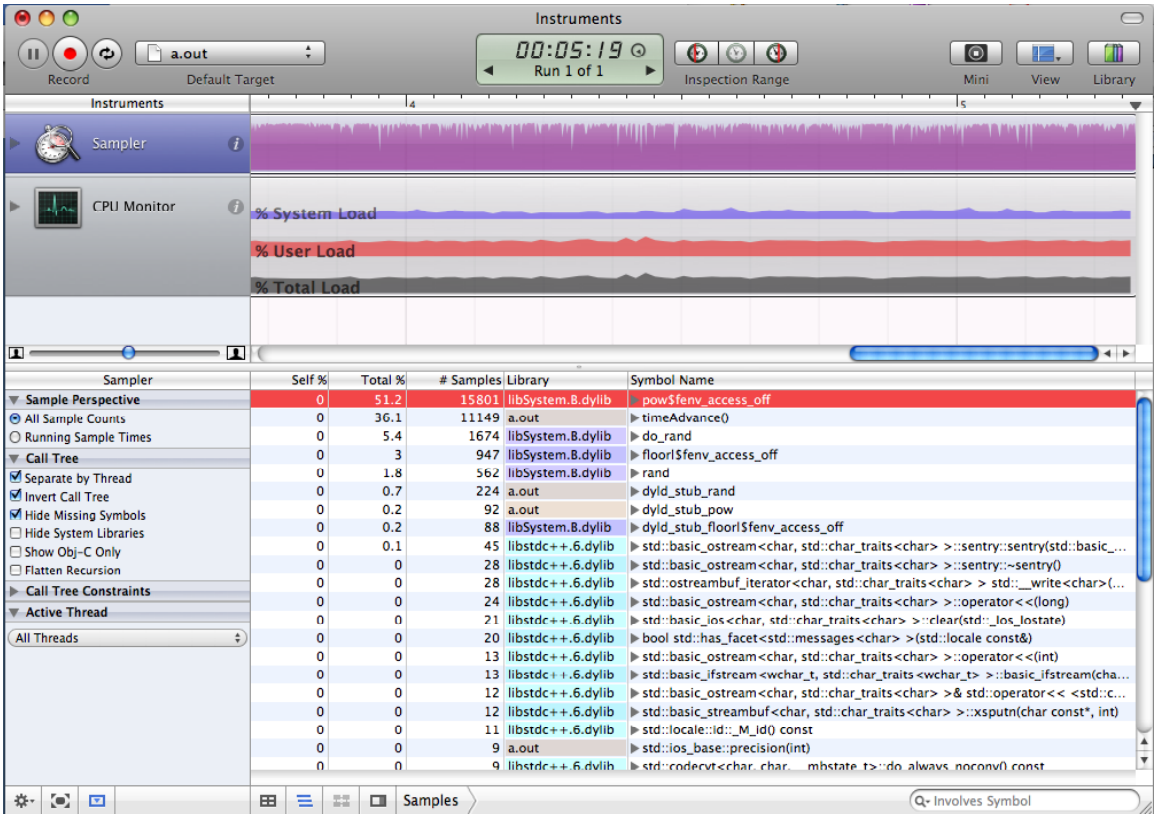

Figure 3: Instruments Software Profiling CASCADE

2.5 Optimization

The first step in optimizing the CASCADE model was to analyze the values associated with the *pow* function call identified during CASCADE profiling. Figure 4 includes three lines of source code associated with the *pow* function call in CASCADE. It can be seen that the `xt2` variable was passed as the power to which the constant variable `FailR` was exponentiated. This implies that if the values assigned to `xt` could be tracked, then the value calculated by `pow(FailR, xt*xt)` could be stored and referenced during program runtime.

```
double xt     = Age[i];
double xt2    = xt*xt;
double test   = pow(FailR, xt2);
```

Figure 4: Calls Associated with *pow* Function in CASCADE

As a starting point, the values assigned to the `xt` variable were tracked, stored, and counted during runtime using a linked list. With each iteration of the for-loop containing the *pow* function, if the value assigned to `xt` had not been seen before, a new linked list node was created. The key for the node was assigned the value of `xt` and the data field contained an integer to be used for counting the number of occurrences of that particular key. If the current `xt` value had been seen previously, the count in the data field was incremented by one.

Compiling and running this analysis version of the CASCADE model quickly revealed that the particular for-loop was executed tens of millions of times during program execution. By printing the linked list nodes after one key was identified ten million times, it was discovered that the `xt` values of 0.0, 1.0, 2.0, and 3.0 all occurred tens of millions of times during a CASCADE model run. Unfortunately, the additional linked list overhead caused the CASCADE model to run out of memory and fail. Fortunately, the program failure did not occur until well into program execution however, therefore the delayed failure allowed the program to complete enough iterations to determine that the four integer values mentioned were frequently common values assigned to the `xt` variable. With this information, it was easy to create a small set of if-else statements that drastically reduced the number of calls to the *pow* function. Figure 5 shows the new optimized source code using if-else statements to replace the previous *pow* calls shown in Figure 4.

```
double xt       = Age[i];
//double xt2    = xt*xt;
double test;
if(xt == 0.0)
{  test = 1.0; }
else if(xt == 1.0)
{  test = FailR; }
else if(xt == 2.0)
{  test = FailR_2; }
else if(xt == 3.0)
{  test = FailR_3; }
else
{  test = pow(FailR,(xt*xt));
}
```

Figure 5: Optimized Source Code in CASCADE

Following initial program start up and the reading of input parameters, the value assigned to the `FailR` variable remained constant throughout the execution of the CASCADE program. This allowed for the `FailR_2` and `FailR_3` variables to be created as static global doubles and assigned the values of `FailR_2 = pow(FailR,4);` and `FailR_3 = pow(FailR,9);` within the *initializeSystem* function. Table 4 shows the timing results from running the CASCADE model with the if-else optimizations. Overall, total execution time dropped by approximately 9% of the original runtime on the Intel Dual Core processor. This difference was also observed on the Sun Opteron processor. To confirm that this optimization would be maintained given different startup conditions, various combinations of input parameters were tested to confirm the consistency of speedup. In summary, the optimizations reducing the total number of calls to the *pow* function in the CASCADE model resulted in a noticeable improvement in code performance.

Table 4: Optimized CASCADE Timing Results

Compiler	**Optimization**	**Architecture**	**Original Time (min)**	**Optimized Time (min)**
g++ 4.0.1	None	2.4 GHz Intel Dual Core, 4GB DDR2 SDRAM	2.63	2.41
g++ 4.0.1	-O3		2.19	1.99
g++ 3.3.3	None	2.6 GHz Sun Opteron Quad Core, 8 GB per socket DDR2-667 RAM	4.95	4.70
g++ 3.3.3	-O3		4.56	4.19
pathCC 3.1	None		2.88	2.80
pathCC 3.1	-O3		2.77	2.73
pgCC 7.2.2	None		2.61	2.44
pgCC 7.2.2	-O3		2.55	2.52

Chapter 3 OPA Model

3.1 Program Description

OPA is a C++ model simulating the dynamics of power transmission networks. After creating a network of power generation nodes interconnected by power lines, the OPA Model simulates power overloads, cascading node and line failures, and power outages on a time based cycle. Each power generation node and power line stores information regarding its particular power load levels. The load levels change and update during each iteration of the model's cycle.

Two timescales are represented in the model, slow and fast. The slow timescale corresponds to a day or year cycle in which power demands change and improvements to the system are made. The fast timescale corresponds to minute or hour cycle in which power overloads, cascading node and line failures, and outages occur. The node and power line failures are then resolved using a Linear Programming Solver to re-distribute power to the operable node and power lines throughout the system. The OPA source code is included in Appendix 3. Figure 6 shows the flow diagram representing the OPA model timescales[4].

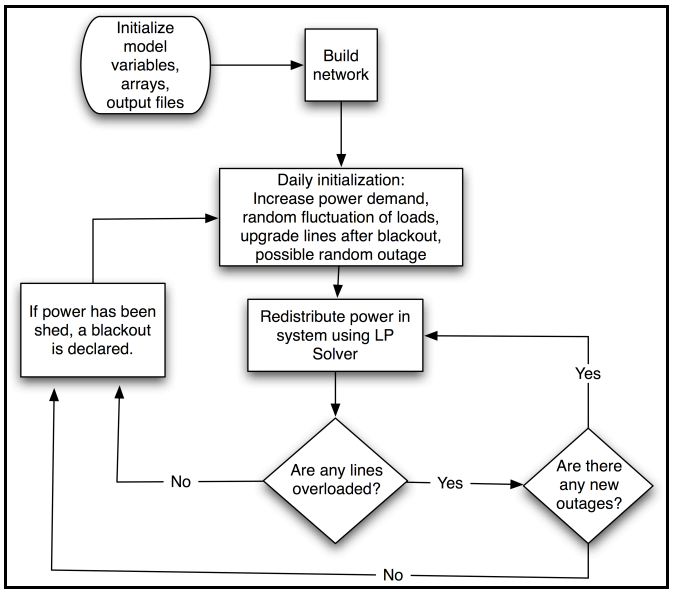

Figure 6: OPA Model Flow Chart

3.2 Benchmarking

The original OPA model benchmarks were obtained by running the program on an IBM p655+ power4 eight processor node with 16 gigabytes of shared memory. This node was a member of the now retired 800 processor IBM system hosted at ARSC. IBM's xlC compiler version 8.0.0.7 was used to compile the code. Different compiler optimization flags were tested to gather many different benchmark results. For program timing, the bash built-in *time* function was used. The *time* function returns clock times to the nearest thousandths of a second, therefore it was a sufficient tool for benchmarking the OPA model. Table 5 shows the summary of timing results for the original OPA code and the compiler flags used to obtain the results. The fastest OPA run was achieved using the `-O3` xlC optimization flag. This optimization decreased program execution by eighteen minutes and forty-seven seconds, or approximately 15%.

Table 5: Benchmarking Results for OPA Model

xlC Flag	Purpose	Time Taken (min:sec)
None	Original code	> 8 hours
qarch=pwr4	Specifies architecture for which code should be generated	Used with each -Ox flag
qtune=pwr4	Specifies architecture for which executable is optimized	Used with each -Ox flag
qstrict	Turns off aggressive optimizations which may change program semantics	Used with -O3 flag
-O0	Quick local optimizations, e.g. constant folding	> 180 min
-O2	Optimizations best for compilation speed and runtime performance	162 : 01
-O3	Some memory and compile-time optimizations in addition to -O2	127 : 58
-O4	Sets architecture, cache, qipa (interprocedural analysis) and qhot optimizations	133 : 10
-O5	Same as -O4 except qipa level set to 2	Failed to compile *
* Produced "Inlining of specified subprogram failed due to the presence of a C++ exception handler" messages.		

3.3 Modularization

The first step taken while working with the OPA model was to improve the source code's readability. By reorganizing the source code, the model's *main* function was restructured from one thousand thirty-four lines to a compact twenty-four lines. Thirteen additional functions were also added during the restructuring.

With the addition of many new functions, it was important to avoid the use of global arrays. The solution to this situation involved passing the reference to the pointer to the start of an array as a function parameter instead of accessing a global array from within the function. The "testArrPointers.cpp" program (included in Appendix 4) is a small test program created to develop and test the passing of references to the pointer to the start of an array among functions. This design worked well for modularizing the OPA code, aided in program readability, helped isolate file input and output, and eased debugging efforts.

3.4 Profiling

Working with the fastest compiler-optimized version of OPA identified during benchmarking, the program was profiled using IBM's xprofiler and gprof software version 4.1.0.0. To use these basic tools, the `-g` and `-pg` flags were added to code compilation, then the program was run. A "gmon.out" file was produced along with the normal OPA output files and contained data for viewing two dimensional images of the profiled program. To view the results of the xprofiler and gprof, the `xprofiler opaModel gmon.out` command was executed in the directory containing the executable and "gmon.out" files.

To interpret the results presented by xprofiler and gprof, an ARSC HPC Newsletter article[1] was consulted for an introduction to xprofiler graphical output. Figure 7 shows function communication within the OPA code using a tree diagram. The tree format was obtained by removing the clustered functions and hiding all library calls.

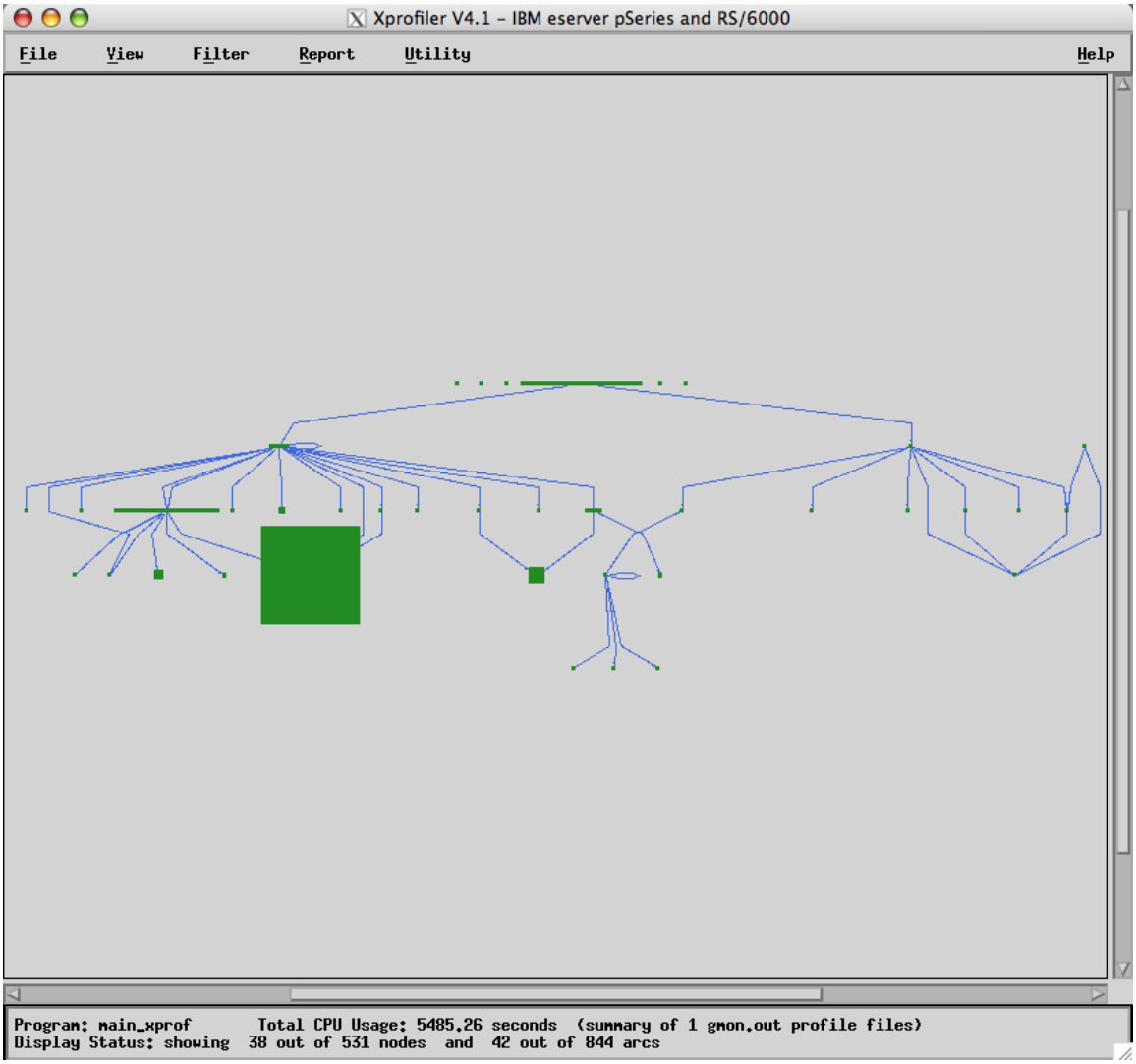

Figure 7: Xprofiler Tree diagram of OPA Function Calls

In Figure 7 each green box represents time spent in one function. The width of the box represents time spent in the function plus time spent in the function's calls to children functions. The box height represents time spent in the one function only. In this case, it is obvious that one very large green box dominated the total OPA computation time. To identify this function and view more details, the Flat Profile was viewed. Figure 8 shows the xprofiler "Flat Profile".

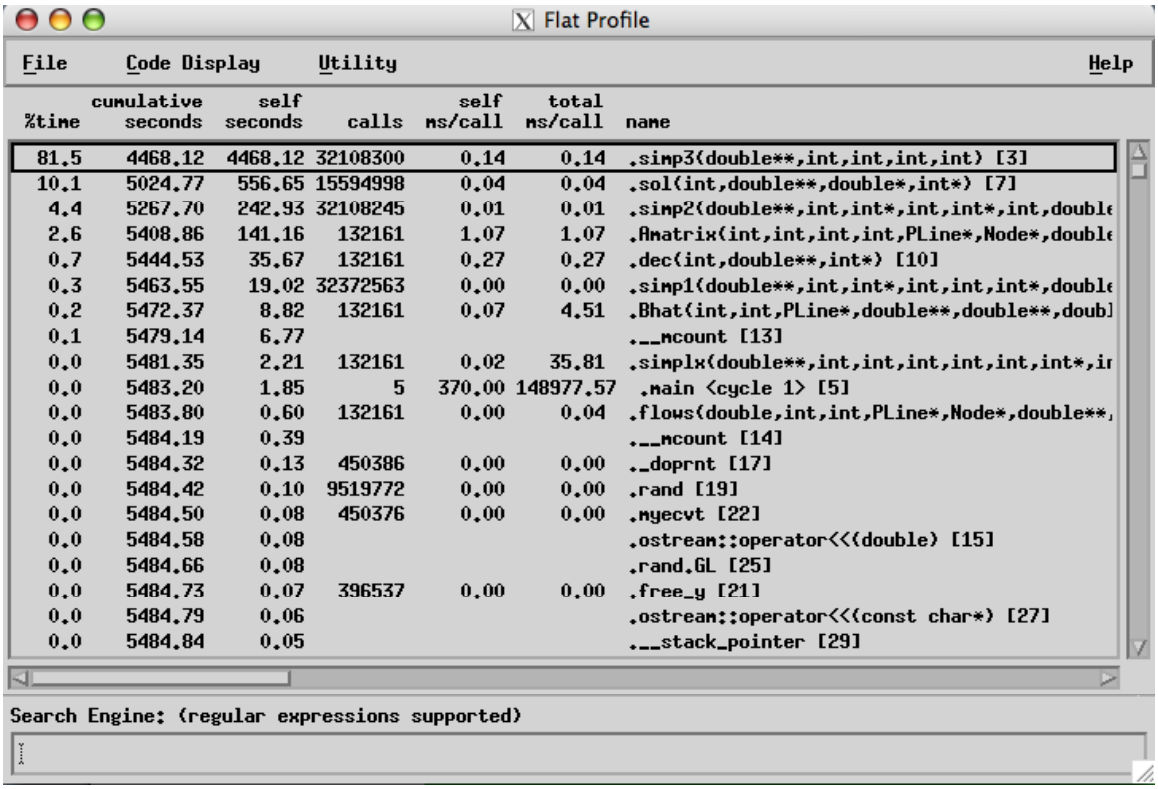

Figure 8: Flat Profile of OPA Generated by xprofiler

The information provided by Figure 8 supports the results shown in the tree diagram in Figure 7. From these results it can be seen that 81.5% of the total OPA execution time was spent in just one function, *simp3*. Therefore, *simp3* became the point of focus for code optimization. Figure 9 shows the contents of the original OPA model *simp3* function.

```
void simp3(double **a,int i1,int k1,int ip,int kp)
{  int kk,ii;
   double piv;
   piv=1.0/a[ip+1][kp+1];
   for (ii=1;ii<=i1+1;ii++)
      if (ii-1 != ip)
      {   a[ii][kp+1] *= piv;
          for (kk=1;kk<=k1+1;kk++)
             if (kk-1 != kp)
                a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
      }
   for (kk=1;kk<=k1+1;kk++)
      if (kk-1 != kp) a[ip+1][kk] *= -piv;
   a[ip+1][kp+1]=piv;
}
```

Figure 9: Contents of OPA *simp3* Function

3.5 Optimization

To simplify testing and avoid running a two or more hour OPA execution with each new revision during the code optimization process, the *simp3* function was extracted from OPA and placed in a web based application used for rapid code development and testing, NetRun[10]. Porting the *simp3* function to NetRun offered a controlled environment in which experimentation with optimization attempts could be conducted easily.

Extracting the *simp3* function required tracing its input parameter values within the OPA code. This proved to be a slight challenge and required the inclusion of print statements to a full scale OPA run. The additional output revealed the values of the 2-dimensional array and four integers being passed to *simp3* during each function call. Slightly different values were sent to *simp3* for each of its five parameters with each function call, but the range in values was small enough to enable the use of a median value to serve the purpose of testing and timing. It was very difficult to determine the exact allocation size of the large array `a`, but this information wound up not being a concern due to the timing limits set in NetRun. Regardless of the size of the array and the initial parameter values, thirty-two million iterations of the *simp3* function (as determined from OPA profiling) far exceeded the maximum two second wall time allowed for NetRun simulations. To overcome this barrier and obtain the much needed timing results, the total number of *simp3* function calls was reduced to thirty-two thousand and the input parameter

values were reduced by a factor of ten. Also, the dimensions of the 2-dimensional array were set to a size small enough to avoid segmentation faults during for-loop indexing. Figure 10 shows the resulting *simp3* function extracted from OPA and modified to run in NetRun.

```
int i1 = 50;
int k1 = 14;
int ip = 47;
int kp = 11;
double a[55][55];
for (int i=0;i<55;i++)
{   for(int j=0; j<55; j++)
    {   a[i][j] = 1;   }
}

for (int z=0; z<32000; z++)
{   int kk,ii;
    double piv;
    piv=1.0/a[ip+1][kp+1];
    for (ii=1;ii<=i1+1;ii++)
       if (ii-1 != ip)
       {
          a[ii][kp+1] *= piv;
          for (kk=1;kk<=k1+1;kk++)
             if (kk-1 != kp)
                a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
       }
    for (kk=1;kk<=k1+1;kk++)
       if (kk-1 != kp) a[ip+1][kk] *= -piv;
    a[ip+1][kp+1]=piv;
}
```

Figure 10: OPA *simp3* Function Modified for use with NetRun

To begin the optimization process, the modified *simp3* function was run several times in NetRun to achieve benchmarking results. Table 6 shows the results from eight runs with both the NetRun optimization flag turned off and on. The speedup gained with the NetRun optimization flag equaled approximately 300%. One hypothesis for this unrealistic speedup is the possibility of if-statement branch prediction within the for-loops.

Table 6: Benchmarking Results of *simp3* Function in NetRun

Original *simp3* in NetRun	**Unoptimized**	**Optimized**
Run #1	400446891.78 ns/call	133324146.27 ns/call
Run #2	402320861.82 ns/call	132179975.51 ns/call
Run #3	406024217.61 ns/call	130579233.17 ns/call
Run #4	402606964.11 ns/call	133574008.94 ns/call
Run #5	395430088.04 ns/call	132026195.53 ns/call
Run #6	395232915.88 ns/call	**129841804.50 ns/call**
Run #7	404049873.35 ns/call	132879018.78 ns/call
Run #8	**394759178.16 ns/call**	130643129.35 ns/call

The first optimization attempt of the *simp3* function involved loop invariant hoisting. At first glance, it can be seen that many `+1` calculations were being executed inside every iteration of each for-loop. This means that with every iteration, a few assembly instructions (e.g. load, add, and store) were generated to calculate this reoccurring `+1`. Given millions of iterations, these extraneous `+1` operations have the potential to add up to a noticeable slow down in program execution time.

A natural optimization solution for the excessive use of the `+1` addition was to calculate the `+1` at the top of the *simp3* function and substitute the new resulting value in the place of the `+1` calculations contained within the remaining function's for-loops. One array indexing value being read during each execution of the inner most `kk` loop was also extracted. The resulting *loop_invariant_hoisting_simp3* function and timing results are shown in Figure 11 and Table 7.

```
int kk,ii;
int ip_1 = ip+1;  // perform the addition outside the for loops
int kp_1 = kp+1;  // dido
int i1_1 = i1+1;  // dido
int k1_1 = k1+1;  // dido
double piv;
piv=1.0/a[ip_1][kp_1];
double temp;
for (ii=1;ii<=i1_1;ii++)
   if (ii-1 != ip)
   {
      a[ii][kp_1] *= piv;
      temp = a[ii][kp_1]; //extract another constant from inner kk loop
      for (kk=1;kk<=k1_1;kk++)
         if (kk-1 != kp)
            a[ii][kk]-=a[ip_1][kk]*temp; // a[ii][kp_1];
   }
for (kk=1;kk<=k1_1;kk++)
   if (kk-1 != kp) a[ip_1][kk] *= -piv;
a[ip_1][kp_1]=piv;
```

Figure 11: OPA Optimization with Loop Invariant Hoisting

Table 7: Benchmarking Results for Loop Invariant Hoisting

Loop Invariant Hoisting	Unoptimized	Optimized
Run #1	340470075.61 ns/call	123390913.01 ns/call
Run #2	354127168.66 ns/call	122302055.36 ns/call
Run #3	351120948.79 ns/call	124744892.12 ns/call
Run #4	348933935.17 ns/call	**118420839.31 ns/call**
Run #5	340147972.11 ns/call	123394012.45 ns/call
Run #6	**340123891.83 ns/call**	120743989.94 ns/call
Run #7	340683937.07 ns/call	122900009.16 ns/call
Run #8	351186990.74 ns/call	119173049.93 ns/call
Fastest Original *simp3*	394759178.16 ns/call	129841804.50 ns/call

Both the unoptimized and optimized versions of the loop invariant hoisting ran faster than the fastest original *simp3* function. The unoptimized version ran 16% faster and the optimized version ran 9.5% faster. These results showed slight improvement, but more optimization techniques needed to be explored before any final conclusions could be drawn.

The second optimization attempt consisted of loop unrolling. Unfortunately, the *simp3* function for-loops were not easily unrolled due to the dependency of the `if (ii-1 != ip)` check. By adding some extra dependency safety checks in the form of additional if-statements, the cases in which the individual for-loops could be unrolled were identifiable. The resulting code and timing results are shown in Figure 12 and Table 8.

```
for (int z=0; z<32000; z++)
{
   int kk,ii;
   double piv;
   piv=1.0/a[ip+1][kp+1];
   for (ii=1;ii<=i1+1;ii+=2)
   {
      if ((ii-1 != ip) && (ii != ip)) // safe to do two ops here
      {
         a[ii][kp+1] *= piv;
         a[ii+1][kp+1] *= piv;
         for (kk=1;kk<=k1+1;kk+=2)
         {  if ((kk-1 != kp) && (kk != kp)) // safe to do two ops here
            {  a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
               a[ii+1][kk]-=a[ip+1][kk]*a[ii+1][kp+1];
               a[ii][kk+1]-=a[ip+1][kk+1]*a[ii][kp+1];
               a[ii+1][kk+1]-=a[ip+1][kk+1]*a[ii+1][kp+1];
            }
            else if (kk-1 != kp)
            {  a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
               a[ii+1][kk]-=a[ip+1][kk]*a[ii+1][kp+1];
            }
         }
      }
      else if (ii-1 != ip)
      {
        a[ii][kp+1] *= piv;
        for (kk=1;kk<=k1+1;kk+=2)
        {  if ((kk-1 != kp) && (kk != kp))
            {  a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
               a[ii][kk+1]-=a[ip+1][kk+1]*a[ii][kp+1];
            }
            else if (kk-1 != kp)
            {    a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
            }
         }
      }
   }
   for (kk=1;kk<=k1+1;kk+=2)
   {
      if ((kk-1 != kp) && (kk != kp))
      {  a[ip+1][kk] *= -piv;
         a[ip+1][kk+1] *= -piv;
      }
      if (kk-1 != kp)
      {  a[ip+1][kk] *= -piv;
      }
    }
    a[ip+1][kp+1]=piv;
}
```

Figure 12: OPA Optimization with Loop Unrolling

Table 8: Benchmarking Results for Loop Unrolling

Loop Unrolling	Unoptimized	Optimized
Run #1	358308076.86 ns/call	91866016.39 ns/call
Run #2	**351260900.50 ns/call**	93410015.11 ns/call
Run #3	351370811.46 ns/call	90467929.84 ns/call
Run #4	358024120.33 ns/call	89696168.90 ns/call
Run #5	359161853.79 ns/call	89661121.37 ns/call
Run #6	359019994.74 ns/call	86969852.45 ns/call
Run #7	362339019.78 ns/call	92691898.35 ns/call
Run #8	357989072.80 ns/call	**87115049.36 ns/call**
Fastest Original *simp3*	394759178.16 ns/call	129841804.50 ns/call

Surprisingly, the loop unrolling technique achieved much faster results than expected. A speedup of 12% for the unoptimized version and 49% for the optimized version was achieved. This large speed up may have been achieved because of the case where the indexing values of `ii` and `kk` actually equal `ip` and `kp` only occurs one time during the entire for loop traversal. This means the loop was able to perform two operations during most iterations. Because of this, an impressive speedup was expected to be achieved.

The third optimization approach consisted of logic substitution (if-statements in place of for-loops.) This optimization technique was targeting the evaluation of `if (ii-1 != ip)` during each iteration of the for-loops. For all of the values in the range `{1,(i1+1)}`, this statement would evaluate to false just one time. Therefore, the code could have been rearranged to avoid this extraneous use of the logic check with each iteration of the for-loop. The solution implemented to avoid the if-statement involved building two for-loops as shown in Figure 13. When the two independent loop indexes were combined, they skipped over the one instance in which the if-statement would have evaluated to true.

```
int kk,ii;
double piv;
piv=1.0/a[ip+1][kp+1];
// for (ii=1;ii<=i1+1;ii++)
for (ii=0;ii<ip;ii++)
{  a[ii+1][kp+1] *= piv;
   //for (kk=1;kk<=k1+1;kk++)
   for (kk=0;kk<kp;kk++)
   {  a[ii+1][kk+1]-=a[ip+1][kk+1]*a[ii+1][kp+1];
   }
   for (kk=kp+1;kk<=k1+1;kk++)
   {  a[ii+1][kk+1]-=a[ip+1][kk+1]*a[ii+1][kp+1];
   }
}
for (ii=ip+1;ii<=i1+1;ii++)
{  a[ii+1][kp+1] *= piv;
   //for (kk=1;kk<=k1+1;kk++)
   for (kk=0;kk<kp;kk++)
   {  a[ii+1][kk+1]-=a[ip+1][kk+1]*a[ii+1][kp+1];
   }
   for (kk=kp+1;kk<=k1+1;kk++)
   {  a[ii+1][kk+1]-=a[ip+1][kk+1]*a[ii+1][kp+1];
   }
}
// for (kk=1;kk<=k1+1;kk++)
for (kk=0;kk<kp;kk++)
{  a[ip+1][kk+1] *= -piv;
}
for (kk=kp+1;kk<=k1+1;kk++)
{  a[ip+1][kk+1] *= -piv;
}
a[ip+1][kp+1]=piv;
```

Figure 13: OPA Optimization with Logic Substitution

Table 9: Benchmarking Results for Logic Substitution

Logic Substitution	Unoptimized	Optimized
Run #1	393122196.20 ns/call	107452869.42 ns/call
Run #2	387072086.33 ns/call	110023021.70 ns/call
Run #3	387042045.59 ns/call	**106110095.98 ns/call**
Run #4	394538164.14 ns/call	110081911.09 ns/call
Run #5	392209053.04 ns/call	109292984.01 ns/call
Run #6	393464803.70 ns/call	106347084.05 ns/call
Run #7	**386831998.83 ns/call**	109618902.21 ns/call
Run #8	393290042.88 ns/call	108555078.51 ns/call
Fastest original *simp3*	394759178.16 ns/call	129841804.50 ns/call

As shown in Table 9, the unoptimized version of the logic substitution produced a speedup of 2% and the optimized version showed 22% speedup. These optimization times support the theory that the execution of two independent for-loops may indeed be faster than following a conditional jump with each iteration of the for-loops.

Overall, the fastest optimization technique tested in the NetRun simulations was loop unrolling. Ironically, after inserting the optimized code modifications for the *simp3* function back into the OPA code and re-running the program, loop unrolling ran as the slowest of the optimized versions. Table 10 shows the timing results of the various optimization techniques added to the *simp3* function within the OPA program.

Table 10: Timing Results for OPA Optimizations

Optimization Technique (using `-O3 -qarch` and `-qtune` flags)	Timing Results
Original Code	127 min 58.881sec
Loop Invariant Hoisting	123 min 41.328 sec
Function Inlining	128 min 26.488 sec
Loop Unrolling	> 240 min

Optimized timing results showed that the loop invariant hoisting optimization approach provided the most speedup for the OPA model. Unfortunately the speedup gained was a very

small 3%. It is a possibility that enabling the compiler's `-O3` optimization flag may have already altered the code in such a way that the three optimization approaches became ineffective.

Following the modularizing, profiling, and optimizing of the OPA model, it was discovered that the version of the code being used was outdated, unused, and referred to as a slow version of the model. This was a bit discouraging because it implied that the work done with this version of OPA did not necessarily carry over to the newer faster version of the model. After obtaining and profiling the newer version of OPA, it was verified that indeed the work done with the slow version was in vain. Attempts to optimize the new version of OPA will be added to the list of future considerations.

Chapter 4 Coupling CASCADE and OPA using REPAST and SWIG

4.1 Description of REPAST and SWIG

One goal of the HSDMAS was to couple CASCADE and OPA together to develop an interactive multi-layer, multi-agent system model. To accomplish this goal, an open source software package was selected to allow CASCADE and OPA to be run simultaneously in a graphical environment supporting interactive control over the models and agents during runtime. A thorough search of available packages on the www.sourceforge.com website led to the selection of the REPAST[11] software version 3.1.

REPAST offered all of the targeted features needed to complete the HSDMAS project, except for one significant design incompatibility: REPAST was written in the Java programming language. To overcome this hurdle, the SWIG[14] sofware version 1.3.31 was used to create Java wrappers from the CASCADE and OPA C++ declarations. Calling the SWIG wrappers within REPAST enables interaction between both models during runtime and introduces little overhead besides the initial programming involved with Java coding.

REPAST is an open source software package intended for modeling agents. It allows for the representation of agents as discrete, self-contained entities in a user designed model, and provides the tools to visualize and interact with the model data in real time. The REPAST software is available for downloading and installs easily. The only trouble encountered during software installation and testing (using the provided examples) consisted of a compilation error message stating `Exception in thread "main" java.lang.NoClassDefFoundError: cern/jet/random/engine/RandomEngine`. A search using the Google Search Engine,, returned a webpage translated from Portugese that contained the exact error message and a resolution to the problem[8]. By recursively copying the *RepastJ/lib* directory into a *RepastJ/lib/lib* subdirectory, the problem was resolved. All subsequent examples and further development using REPAST worked as expected.

The SWIG software package takes C or C++ declarations and creates wrappers to be accessed by Java code. This wrapping technique provided a transparent interface for Java and C/C++, which enabled the REPAST code to call functions and update variable values within CASCADE and OPA during runtime.

Downloading and installing SWIG was straightforward, although testing the software required the creation of a specialized makefile to link against the appropriate SWIG libraries. Once the SWIG provided test cases were successfully compiled and executed, the working "Makefile" (found in Appendix 5) could be used as a template for the compilation of the HSDMAS project code.

4.2 Model Design

The first step in coupling CASCADE and OPA involved generating the Java wrappers using SWIG. This implies that the CASCADE and OPA models communicate with REPAST through SWIG, as represented in Figure 14. The resulting multi-layered model was initialized within REPAST and run as a Java executable. The utility agents representing human factors were also hosted in REPAST and interact and adapt with the running CASCADE and OPA models.

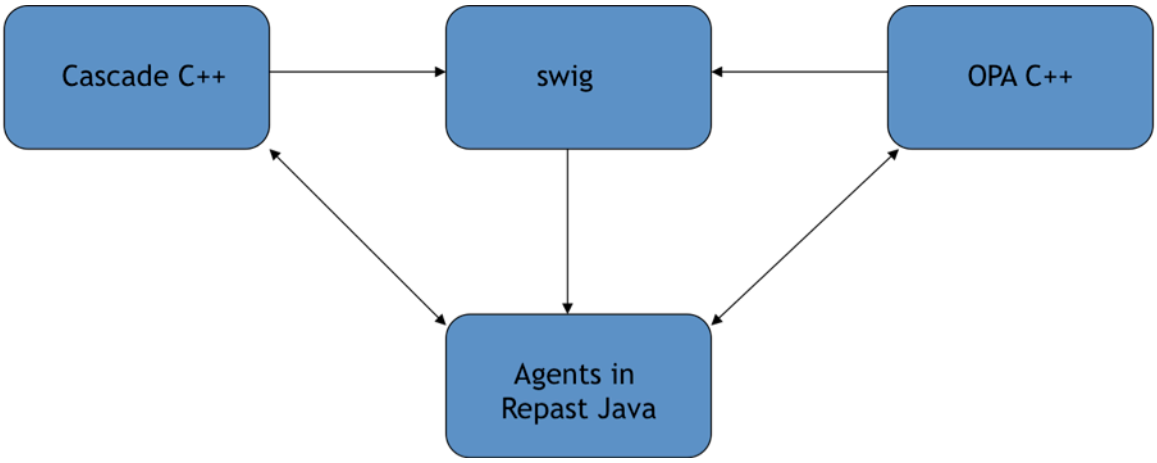

Figure 14: Model of Communication in HSDMAS

The first step taken to enable this model coupling required the creation of a SWIG wrapper file containing the C++ functions and variables accessed by REPAST. The content of these files follows specific formatting requirements to enable compatibility with the SWIG software. The two wrapper files created for the HSDMAS are “cascadeModel.i” and “opa.i” and are included in Appendices 6 and 7. All functions and variables included in these files are the functions and variables accessed by the Java REPAST code. In most cases, the CASCADE and

OPA C++ variables were declared with a global scope, and *get* and *set* functions were created to allow REPAST to retrieve and set the variables in real time as the C++ programs executed their code.

The second step in model coupling involved the updating of program control. This REPAST based model is controlled by the main REPAST Java class, therefore the CASCADE and OPA programs needed to be rewritten. Originally stand alone models, CASCADE and OPA were updated to be run as a series of function calls instead of be executed with a call from their *main* functions.

4.2.1 Using Macros and a Makefile

This significant reorganization of code introduced potential difficulties with model output validation and verification. To preserve the original CASCADE and OPA functionality in addition to allowing compatibility with the REPAST model, C++ Macros were used to enable compilation of either form of the program: the original version or the REPAST compatible version. This technique worked very well and allowed for easy compilation of either version using one makefile. The Macro controlled CASCADE and OPA C++ code is included in Appendices 1 and 3. Appendix 5 contains the makefile used to compile both the original and REPAST compatible versions of the two models.

4.3 REPAST Code Development

After the CASCADE and OPA models were updated to be compatible with the REPAST program, the Java code was written to build and initialize the REPAST model from the incorporated CASCADE and OPA functions and variables. The first Java code included the loading of the new Java libraries created from the SWIG wrapper files. These libraries were loaded within the REPAST Java program by adding the commands shown in Figure 15 to the main REPAST source code file found in Appendix 8.

```
System.loadLibrary("cascadeModel");
System.loadLibrary("opa");
```

Figure 15: CASCADE and OPA Libraries Loaded in REPAST

The next step in coupling the CASCADE and OPA models included the creation of the main *step* function in the REPAST Java file. This *step* function contains the code to be executed for each timestep, or iteration, of the REPAST model. Creating this function was quite laborious and involved the integration of the REPAST graphical functions with the flow of control from both the CASCADE and the OPA models.

Using the CASCADE and OPA flowcharts as shown in Figures 1 and 6, the REPAST *step* function was written to execute the CASCADE and OPA model iterations. Within each REPAST timestep, the appropriate Java library (SWIG wrapper) function calls were programmed to call the CASCADE and OPA functions and variables needed to execute the model iterations. Any graphical information displayed in REPAST was also updated within the *step* function. Appendix 8 includes the Java REPAST source code in the file "Cascade.java".

Following much experimentation, it was determined that for each REPAST timestep, five thousand iterations of the CASCADE model and five hundred iterations of the OPA model were called. Without these micro timesteps, the slowdown from updating the graphical representation of variable data was tremendous and drastically hindered the performance of the overall model.

4.4 Integrating Agents with REPAST

Following the integration of the CASCADE and OPA models with REPAST, an additional goal of the HSDMAS was to integrate independent learning agents with the model. It was quite obvious that extracting the existing agents from CASCADE and OPA then implementing them in the REPAST model would be the best place to begin agent integration. This agent extraction involved some code conversion from C++ to Java including the use of the random number libraries. Verification that the agent functions returned equal values from both versions was accomplished by temporarily calling the C++ random number generator from both C++ and Java programs.

Much time was spent validating the correctness of the ported agent functions, primarily due to a programming error of failing to update all related variables within the agent functions. As visible within the "Cascade.java" file, several SWIG wrapper *set* functions were executed just before the agent functions return. Failing to update these variable values during model execution caused an inconsistency in agent data output. This problem was eventually resolved when the error was discovered and corrected.

With the existing agents successfully extracted and programmed to execute independently from the CASCADE and OPA models, the HSDMAS took form. This resulting multi-agent system allows the user to interact with and visualize the coupled CASCADE and OPA models in real-time.

Figures 16 and 17 show screen shots of two of the CASCADE and OPA agents graphed in real-time as the HSDMAS runs in REPAST. Note that the running program can be paused at any time to update particular agent or model parameter values via the REPAST Control Bar and the REPAST Input Parameters Menu shown in Figures 18 and 19.

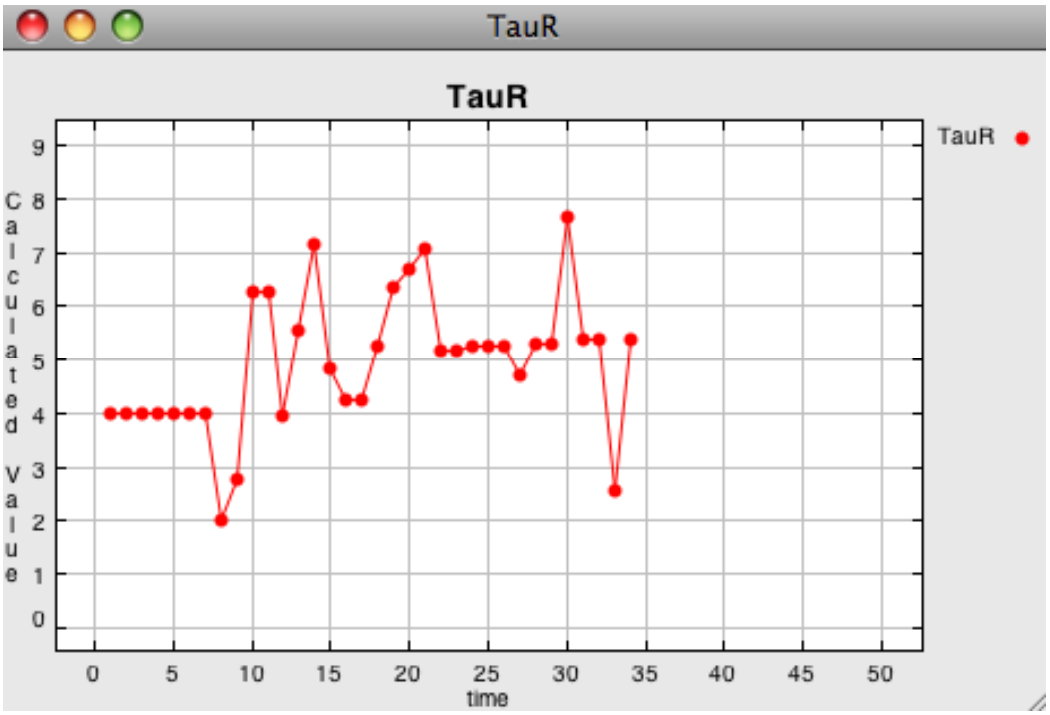

Figure 16: REPAST `TauR` Agent Graphed in Real Time

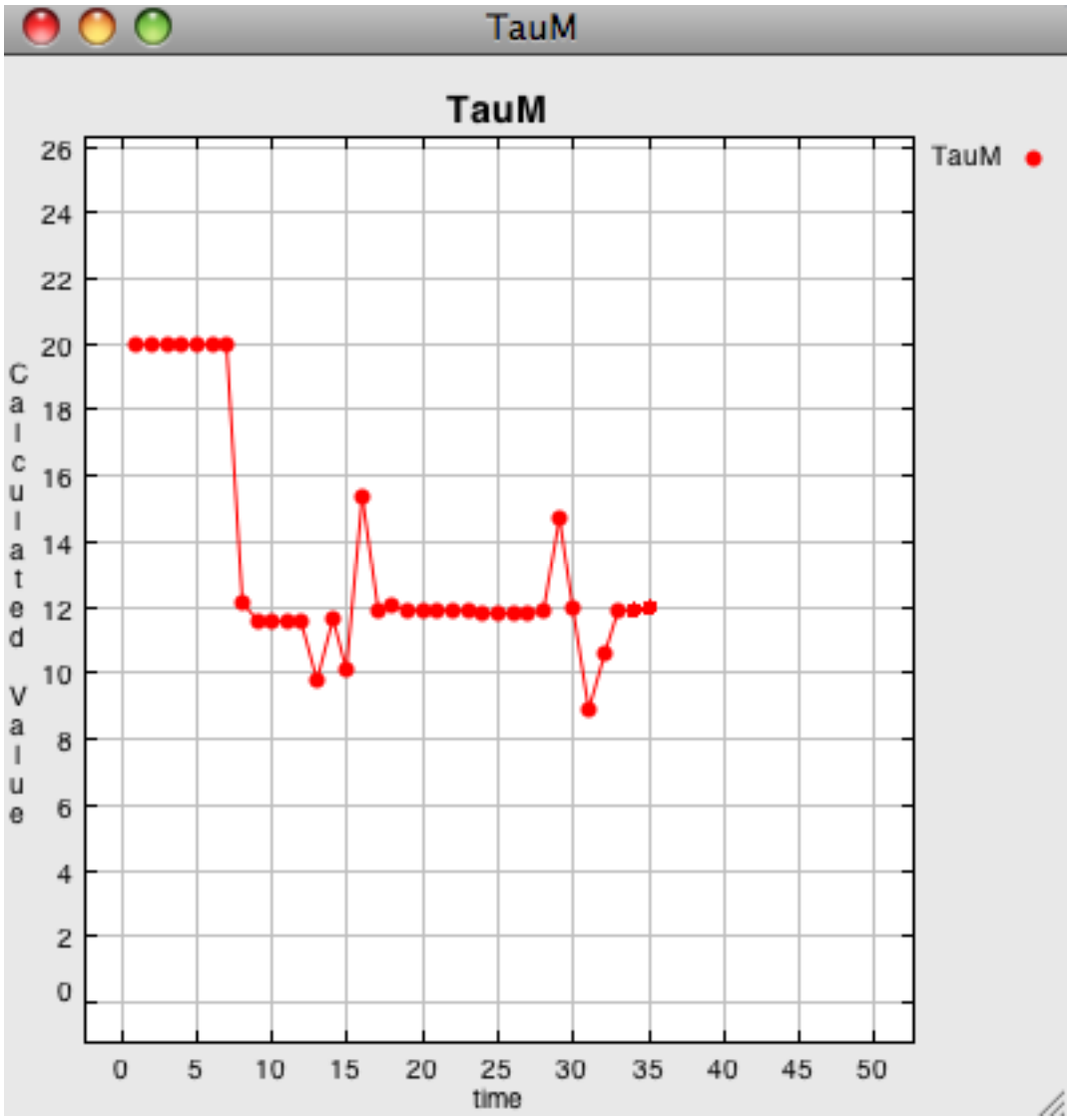

Figure 17: REPAST `TauM` Agent Graphed in Real Time

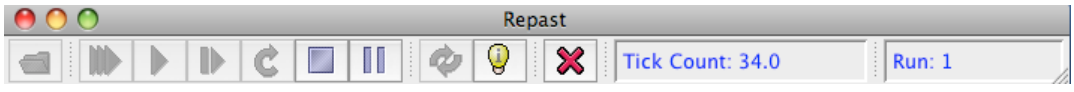

Figure 18: REPAST Control Bar

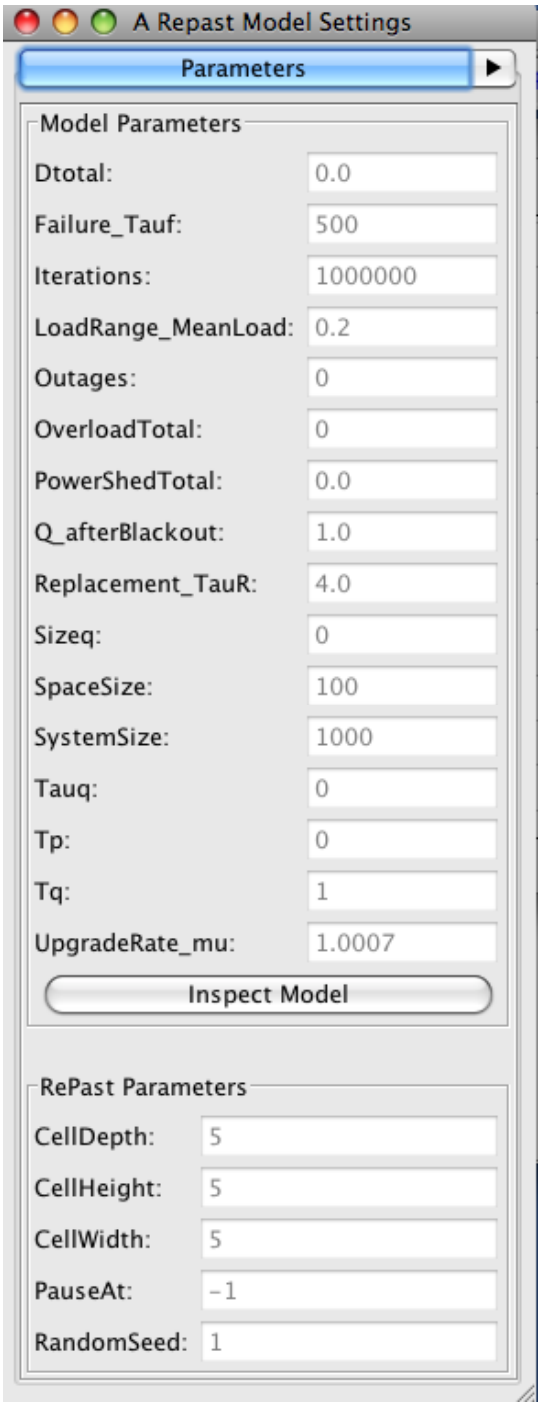

Figure 19: REPAST Input Parameters Menu

4.5 Verification and Validation

The final step in coupling CASCADE and OPA with REPAST was to verify that the output produced during original CASCADE and OPA runs matched the REPAST output exactly. Verification was accomplished by compiling both versions using the implemented C++ Macros, then comparing the output data produced. One problem encountered with this technique was the slight difference in values generated by the C++ and Java random number libraries. To overcome this problem, a C++ function was added to the CASCADE and OPA models which allowed REPAST to call the C++ random number library and use values produced from the C++ random number generator instead of the Java random number generator. This process worked well and produced matching output data and files.

Table 11 shows the timing results from the original model runs and the coupled model runs using REPAST and SWIG. There is a three hundred minute slowdown when running the HSDMAS compared to running OPA independently. This timing difference is quite surprising because running the CASCADE model individually from within REPAST created less than a 10% slowdown in total execution time.

Table 11: Timing Results for Models

Model	Compiler, Processor	Time Taken (minutes)
CASCADE	g++ -O3, 2.4 GHz Intel Dual Core	1.42
OPA	g++ -O3, 2.4 GHz Intel Dual Core	69.63
HSDMAS	g++ -O3 and javac, 2.4 GHz Intel Dual Core	369.45

No logical cause of program slowdown was identified from looking through the code, therefore continued research of the slowdown led to two different profiling tools: Instruments[6]version 1.5 and Shark[13] version 4.6.0. Both of these commercial off the shelf software packages are advertised to profile running processes on the Mac Leopard Operating System. Unfortunately neither was able to successfully attach their monitoring tools to the original OPA C++ model or the REPAST Java program. Communication with the Instruments

developers has begun to determine the cause of the software's inability to profile a user owned C++ program. At this time no further information is known. Pursuing and resolving the cause of the unacceptable slowdown is a very high priority, but is beyond the scope of this HSDMAS project. The continuation of tracking down the cause of the REPAST slowdown will be added to the future considerations of the HSDMAS.

Chapter 5 Conclusion

5.1 Summary

The goals of the HSDMAS project were to improve and optimize the performance of the CASCADE probabilistic model, to improve and optimize the performance of the dynamic power systems model OPA, and to couple CASCADE and OPA together to develop an interactive multi-layer, multi-agent system modeling power transmission and communication. Following CASCADE model profiling and optimization, program speedup was achieved and executed 9% faster than the original model runtime. This was accomplished by simply replacing the *pow* math library call with if-else statements. OPA profiling and optimization resulted in a 3% speedup using the loop invariant hoisting optimization technique. Unfortunately the version of the OPA model profiled and optimized was obsolete, therefore profiling and optimization efforts must be repeated on the current working version of OPA. Finally, the CASCADE and OPA models were coupled using the SWIG and REPAST software. A graphical user interface was implemented to allow interactive control and visualization of the coupled models and associated independent learning agents, creating the HSDMAS. Based on these results, the HSDMAS should be a valuable contribution to the Human Decision Making Dynamics and its Impact on Infrastructure Systems research.

5.2 Future Considerations

Future considerations for the HSDMAS include work that lies beyond the scope described for this project. The first consideration should begin with profiling and optimizing the newer version of the OPA C++ code. If the newer version can be sped up, then the entire HSDMAS model execution time will also be optimized.

A second consideration should be finding a tool to profile the Java based HSDMAS as it runs, then optimizing the very slow REPAST code while it runs the OPA model.

A newer version of REPAST was released in 2008 and is now called REPAST SIMPHONY[12] version 1.2.0. This newer version includes more functionality available through the REPAST GUI and better visualization tools. Updating the HSDMAS code to be REPAST SIMPHONY compatible offers the potential for a more versatile and improved interactive approach to visualizing the HSDMAS model as it runs and should therefore be a third consideration.

Finally, the implementation and testing of new learning agent types will be a worthwhile fourth consideration to potentially increase the value and versatility of the HSDMAS. The REPAST software includes several intelligent agent examples such as genetic, neural, and regression learning algorithms. Adding these existing agent types to the CASCADE and OPA models would be an excellent test of the effectiveness of the existing models.

5.3 Lessons Learned

Throughout the duration of the project, many versions of working and non-working code were created. Instead of copying the files to a backup location, it would have been practical to use version control software. Unfortunately version control software was not used for a majority of this project, although the data was backed up often by burning the files to cd and emailing copies to an email address, therefore storing the files on the email server.

When working with new data structures or unfamiliar conditions such as references to pointers as function parameters, it is very helpful to creature miniature programs which replicate the new idea or concept. This technique proved invaluable while modularizing OPA (as mentioned in section 3.3.) It was very easy to test the passing of references to pointers to the start of arrays among functions in a small test program instead of trying to implement the design in the large production program.

It is always good to take notes throughout the entire duration of a large project, even if it is simply a sentence or two each day. This creates a written document showing progress made. The document can also be referenced in the future while writing a summary or project write up.

Finally, the most important lesson learned is to start early. There are always unexpected events to deal with. Dealing with these events takes time, therefore beginning work early and working consistently helps avoid unnecessary stress and anxiety.

References

1. "ARSC HPC Users' Newsletter 275". Arctic Region Supercomputing Center. Referenced March 2007. <http://www.arsc.edu/support/news/HPCnews/HPCnews275.shtml#article1>
2. Carreras, B.A. “A Dynamical Cascading Model that may be used to Incorporate Agent Operators.” June 2007. Available from bacv@comcast.net
3. Carreras, B.A., Lynch, V.E., Sachtjen, M.L., Dobson, I., Newman, D.E. "Modeling Blackout Dynamics in Power Transmission Networks with Simple Structure." Hawaii International Conference on System Sciences. Jan. 2001.
4. Carreras, B. A., Newman, D. E., Dobson, I., Zeidenberg, M. "The Impact of Risk-Averse Operation on the Likelihood of Extreme Events in a Simple Model of Infrastructure". Submitted to Physical Review E. (2007)
5. Dobson, I., Carreras, B., Lynch, V., Newman, D. “An Initial Model for Complex Dynamics in Electric Power Systems Blackouts.” Hawaii International Conference on Science Systems. Jan. 2001.
6. "Instruments", version 1.5. Available with the Macintosh Leopard Operating System, version 10.5.6. Apple Inc. <www.apple.com>.
7. "Manifesto on the California Electricity Crisis" 2001. University of California, Berkeley. Referenced December 2008. <http://faculty.haas.berkeley.edu/spiller/eleabs.htm#Manifesto%20on%20the%20California%20Electricity%20Crisis>
8. "Matematica - Modelo Bak-Sneppen" Referenced July 2007. <http://translate.google.com/translate?hl=en&sl=pt&u=http://www.nabble.com/Matem%25C3%25A1tica---Modelo-Bak-Sneppen-t3177357.html&sa=X&oi=translate&resnum=1&ct=result&prev=/search%3Fq%3Duchicago.src.sim.engine.SimModelImpl.%253Cinit%253E(Unknown%2BSource)%26hl%3Den>
9. Newman, D. E., Nance, K. L., Dobson, I., Zeidenberg, M. “DHB Collaborative Research: Human Decision Making Dynamics and its Impact on Infrastructure Systems” National Science Foundation SBE - HSD - Dynamics of Human Behavior Proposals. (NSF-SES-062361) Feb 2006.

10. Lawlor, Orion. "NetRun". University of Alaska Fairbanks. Referenced February 2008. <http://lawlor.cs.uaf.edu/NetRun/help.html>
11. “Recursive Porous Agent Simulation Toolkit" (REPAST), version 3.1. Argonne National Laboratory. Referenced July 2007. <http://repast.sourceforge.net>.
12. "REPAST SIMPHONY", version 1.2.0. Argonne National Laboratory. Referenced December 2008. <http://repast.sourceforge.net>.
13. "Shark", version 4.6.0. Available with the Macintosh Operating System. Apple Inc. <www.apple.com>
14. "Simplified Wrapper and Interface Generator" (SWIG), version 1.3.31. Referenced September 2007 <http://www.swig.org>.
15. "Storm leaves at least 1 million without power in Northeast". CNN.com. Referenced December 2008. <http://www.cnn.com/2008/US/weather/12/12/storm.new.york.massachusetts/index.html>
16. "TAU Portable Profiling Package". University of Oregon. Referenced November 2008. <http://www.cs.uoregon.edu/research/tau>

Appendix 1: CASCADE Model Source Code: "cascadeModel.cpp"

```
// Failure2.8.5.cp Model to study failures in a system
// Started May 2005 in San Augustin, Florida
// Distinction between time of replacement and effective age of a
// component
// Age of a component made a double 7/13/05
// Version 2.4, 12-9-2006
// Made Load and aging as two separate causes
// for failure. Incorporated cascading for load
// Version 2.5, 12-16-2006
// Included growth of load demand and
// upgrade of the components like OPA
// Version 2.6 Self-organization , 3-22-2007
// Version 2.7.1 Regulates maintenance time to increase benefits
// Adding risk aversion
// Version 2.8.0 Modularization of the code
// Version 2.8.1 Second agent
// Version 2.8.2 risk aversion as a function of time
// 4/6/2007
// Version 2.8.3 modification of utility function
// quadratic term in mu-1.    7/24/2007
// Version 2.8.4 risk taking option as a function of time
// 8/11/2007
// Version 2.8.5 splitting maintenance and replacement.
// It is a 3 agent version 11/1/07
// Dec 2008 ONN Modularized and optimized for use with REPAST

#include <iostream>
#include <fstream>
#include <iosfwd>
#include <math.h>
#include <stdlib.h>
#include <time.h>      // for clock(), CLOCKS_PER_SEC
using namespace std;

/* Globals */

double g       = RAND_MAX;

/* Globals used for swig */
// Agent1
double * MP;
double * TRA;
int AK;

//Agent2
double pt_MaxP2;
double pt_Maxmu;
double * MP2;
double * MU;
int pt_Ak2;
int pt_kk2j;
double pt_gamma2;
/* static variables used for program optimization */
static double FailR_2;
static double FailR_3;


/* more globals to enable function calls from java */
int Iterations                    // Number of days
int Size              = 1000;   // System size (number components)
int j          = 0;              // Iteration index
long Tauf             = 100;    // Averaged lifetime of components
double TauR    = 50;
double TauM    = 10;            // Averaged maintenance time
double TauD    = 50;
double WaitTime       = 0;      // Waiting time between failures of a component
```

```
double p              = 1;          // Added stress on other components
double p0             = 2;          // Added stress on other components
double Factor1        = 0.0;        // Effect of loading on aging
double Factor2        = 0.0;        // Effect of aging on maximum load
double MaxLoadN       = 5000;       // Maximum load
double FailLoadN      = 5000;       // Failing load
double MeanLoad       = 4000;       // Averaged line load at t=0
double LoadRange      = 0.20;       // Range in load variation normalized
int kc         = 0;                 // Printing index
int kmax              = 10;         // Frequency of print
double gamma2     = 1.00005;    // rate of growth of load demand
double mu         = 1.0007;     // Rate of upgrade of the lines
int Duration      = 0;          // Cascade duration
int Ktrans        = 100;        // Maximum number of components where
                                // to which power is transferred from
                                // a failing component
int Jevaluation   = 100;        // Number of days over which evaluation
                                // of benefits is done
double Profit     = 0.0;        // Daily profit
double MeanProfit = 0.0;        // Mean daily profit Agent 1
double MeanProfit2 = 0.0;       // Mean daily profit Agent 2
double MeanProfit3 = 0.0;       // Mean daily profit Agent 2
int je;                         // Daily index during evaluation
double XJ         = Jevaluation;
double MaxT       = 0.0;
double meanP      = 0.0;
double meanM      = 0.0;
double XPiC2      = 0.0;
double XXmm          = 0.0;
double XXMMT   = 0.0;
double XXMMT3  = 0.0;
double meanP2     = 0.0;
double meanP3     = 0.0;
double Profit2    = 0.0;
double Profit3    = 0.0;
int kkj       = 1;
int kkj3              = 1;
int ijk        = 0;
int ijk3              = 0;
int ijt        = 0.;
int ijt3              = 0.;
int jthr              = 30000;
int jlast             = 0;
int Ak         = 0;
int Ak2        = 0;
int Ak3        = 0;
int kk2j          = 1;
double q;
double q0;                      // q value after a blackout
double Tauq;                    // Time to forget a blackout
double Sizeq;                   // Threshold for blackout size
int iq;
int j00           = 0;
double Xprofit    = 0.0;
double Utilization = 0.0;
double MeanMax = 0.0;
double MeanMaxT       = 0.0;
double MeanMaxTm      = 0.0;
int NCascade1  = 0;
int NCascade2  = 0;
int NCascade1T = 0;
int NCascade2T = 0;
double XXk            = 0.0;
double XXj            = 0.0;
double Cost           = 0.0;
double meanCost       = 0.0;
double meanCostT      = 0.0;
double kappa   = 0.8;
int  jkk          = 0;
```

```
long *LastFail;
long *Component;
long *Time;
long *TimeM;
double *Age;
double *Load;
double *MaxLoad;
double *FailLoad;
double *MaxP2;
double *MP3           = new double[20000];  // Monthly averaged daily
                                                // profit for agent 3
double *TRM;                          // Maintenance time
double *Maxmu;
double *Maxday;
double *MeanOrder;
double *MeanMaxP2;
double *MaxP;
double *MaxTRA;
double *MaxdayT;
double *MeanMaxP;
double *MeanMaxP3;
double *MaxP3;
double *MaxTRM;
double *MaxdayTm;
double XTauf2;
double FailR;
double meanF;
double meanR;
double cost;
double meanMu         = 0.;
double meanTauR               = 0.;
double meanTauM               = 0.;
double CountCas               = 0.;
double CountLarge             = 0.0;
double meanD          = 0.0;
double qmid                   = (1. +q0)/2.;
double PiC2                   = 0.0;

std::ofstream fout1("Failures");
std::ofstream fout2("Cascades");
std::ofstream fout3("Time Evolution");
std::ofstream fout4("AgeDistribution");
std::ofstream fout5("Agent-1");
std::ofstream fout6("Agent-2");
std::ofstream fout7("TimeSeries");
std::ofstream fout("WaitTimes");
std::ofstream fout8("Cascades-q-low");
std::ofstream fout9("Cascades-q-highNonAdverse");
std::ofstream fout10("SummaryData");
std::ofstream fout11("Agent-3");
clock_t start_time;
clock_t stop_time;

void initializeSystem();
void timeAdvance();
void beforeAgents(int jk);
void afterAgents(int jk);
void updateProfits();
int closeFiles();

#ifdef REPAST

/* For simulating the C rand # generator */
int getNextRand();
double getRAND_MAX();

/* Functions used by Agent1 */
double getMaxPValue(int);
```

```
void setMaxPValue(int, double);
double getMaxTRAValue(int);
void setMaxTRAValue(int, double);
double getMaxdayTValue(int);
void setMaxdayTValue(int, double);
double getMPValue(int);
void setMPValue(int, double);
double getTRAValue(int);
void setTRAValue(int, double);

/* Functions used by Agent2 */
double getMaxP2Value(int);
void setMaxP2Value(int, double);
double getMeanMaxP2Value(int);
void setMeanMaxP2Value(int, double);
double getMaxmuValue(int);
void setMaxmuValue(int, double);
double getMaxdayValue(int);
void setMaxdayValue(int, double);
double getMP2Value(int);
void setMP2Value(int, double);
double getMUValue(int);
void setMUValue(int, double);
double getPt_MeanMax(int);
void setPt_MeanMax(int, double);

/* Functions used by Agent3 */
double getMaxP3Value(int);
void setMaxP3Value(int, double);
double getMeanMaxP3Value(int);
void setMeanMaxP3Value(int, double);
double getMaxTRMValue(int);
void setMaxTRMValue(int, double);
double getMaxdayTmValue(int);
void setMaxdayTmValue(int, double);
double getMP3Value(int);
void setMP3Value(int, double);
double getTRMValue(int);
void setTRMValue(int, double);

#else

// Functions prototypes
// Agent 1 taking care of maintaining the system
double Agent1(int Jevaluation, double *MaxP,double *MeanMaxP, double *MaxTRA, double
*MaxdayT, double *MP,double *TRA, double *pt_MeanMaxT, double *pt_XXMMT, int *pt_Ak,int
j, int jk, int jkk, int *pt_kkj, int *pt_ijt,double TauR);

// Agent 3 replacement components
double Agent3(int Jevaluation, double *MaxP,double *MeanMaxP, double *MaxTRA, double
*MaxdayT, double *MP,double *TRA, double *pt_MeanMaxT, double *pt_XXMMT, int *pt_Ak,int
j, int jk, int jkk, int *pt_kkj, int *pt_ijt,double TauR);

//Agent 2 upgrading components
double Agent2(int Jevaluation, double *MaxP2,double *MeanMaxP2, double *Maxmu, double
*Maxday,double *MP2,double *MU, double *pt_MeanMax, double *pt_XXmm, int *pt_Ak2,int j,
int jk,int jkk, int *pt_ijk, double *pt_gamma, double mu );
```

```
// main program
int main (int argc, char * const argv[]) {
  int jk = 0;                    // Monthly index
  initializeSystem();
  while(j < Iterations)
  {   timeAdvance();
      if(j > jthr)
      {
         je++;
         if ( je > Jevaluation-1)
         {
            beforeAgents(jk);
            TauR  = Agent1(Jevaluation, MaxP, MeanMaxP,MaxTRA,MaxdayT, MP, TRA,
&MeanMaxT,&XXMMT,&Ak, j, jk, jkk,&kkj,&ijt, TauR);
            mu = Agent2( Jevaluation,MaxP2,MeanMaxP2,Maxmu,Maxday,MP2, MU,
&MeanMax,&XXmm,&Ak2, j, jk,jkk,&ijk,&gamma2, mu);
            TauM = Agent3(Jevaluation, MaxP3, MeanMaxP3, MaxTRM, MaxdayTm, MP3, TRM,
&MeanMaxTm, &XXMMT3, &Ak3, j, jk,jkk, &kkj3, &ijt3, TauM);
            afterAgents(jk);
            meanCost = 0.0;
            jk++;
            jkk++;
         }
         else
         {  updateProfits(); }
      }
      j++;
      kc++;
   }
   closeFiles();
   exit(0);
}
#endif

double getRAND_MAX()
{   return (double) RAND_MAX; }

int getNextRand()
{   return rand(); }

void initializeSystem()
{
      Iterations  = 1000000;
        Jevaluation    = 100;  // Number of days over which evaluation of
                           // benefits is done
        int SizeDim    = Iterations/Jevaluation; // Total number of
                                                 // evaluation periods
        je             = 0;      // Daily index during evaluation
        q              = 1.0;
        q0             = 1.0;  // q value after a blackout
        Tauq           = 1000; // Time to forget a blackout
        Sizeq          = 100;  // Threshold for blackout size
        iq             = 0;
        kappa          = 0.8;

        start_time = clock();

        /* initialize random seed: */
        srand ( time(NULL) );

        //input time evolution parameters

        std::cout      << "\n\t Failure model version 2.8.5 ";
        std::cout      << "\n\tNumber of time steps: ";
        std::cin              >> Iterations;
      std::cout       << "Iterations = " << Iterations <<"\n";
      std::cout       << "\n\tFailure time =   ";
        std::cin              >> Tauf;
      std::cout       << "Tauf = " << Tauf << "\n";
```

```
std::cout       << "\n\tReplacement time =   ";
 std::cin             >> TauR;
std::cout       << "TauR = " << TauR << "\n";
std::cout       << "\n\tMaintenance time =   ";
 std::cin             >> TauM;
std::cout       << "TauM = " << TauM << "\n";
std::cout       << "\n\tupgrade rate mu =   ";
 std::cin             >> mu;
std::cout       << "mu = " << mu << "\n";
 //std::cout    << "\n\tLoad range/Mean Load =   ";
 //std::cin             >> LoadRange;
//std::cout       << "LoadRange = " << LoadRange << "\n";
//std::cout     << "\n\tp =   ";
 //std::cin             >> p0;
//std::cout       << "p0 = " << p0 << "\n";
 //std::cout    << "\n\tSystem size =   ";
 //std::cin     >> Size;
//std::cout       << "Size = " << Size << "\n";
 std::cout      << "\n";
 std::cout<<"*********************************************\n";
 std::cout      << "*
                                          *\n";
 std::cout      << "*  Choice of agent behavior (iq):
                *\n";
 std::cout      << "*
                                          *\n";
 std::cout      << "*  1) Constant q
                        *\n";
 std::cout      << "*  2) Risk averse reaction to large events
        *\n";
 std::cout      << "*  3) Risk taking reaction to quiet periods
        *\n";
 std::cout      << "*
                                          *\n";
 std::cout      << "*
                                          *\n";
 std::cout      << "*
                                          *\n";
 std::cout<<"*********************************************\n\n";
 std::cout       << "iq = ";
 std::cin       >> iq;
 std::cout       << "iq = " << iq << "\n";
 switch (iq) {
               case 1:
               std::cout << "\n\t What is the value of q =   ";
               std::cin  >> q;
          std::cout << "q = " << q << "\n";
               break;

               case 2:
               std::cout << "\n\t q after blackout q0 =   ";
               std::cin  >> q0;
          std::cout << "q0 = " << q0 << "\n";
               std::cout <<"\n\tTime to forget a blackout Tauq= ";
               std::cin  >> Tauq;
          std::cout << "Tauq = " << Tauq << "\n";
               std::cout <<"\n\tThreshold for blackout size Sizeq=";
               std::cin  >> Sizeq;
          std::cout << "Sizeq = " << Sizeq << "\n";
               break;

               case 3:
               std::cout <<"\n\tq after after quiet period q0 =   ";
               std::cin  >> q0;
          std::cout << "q0 = " << q0 << "\n";
 std::cout << "\n\t Time to begin going to risk taking operation Tauq =   ";
               std::cin  >> Tauq;
          std::cout << "Tauq = " << Tauq << "\n";
               std::cout << "\n\t Decay time to risk taking operation TauD =   ";
```

```
                    std::cin  >> TauD;
                std::cout << "TauD = " << TauD << "\n";
                    std::cout << "\n\tThreshold for blackout Sizeq= ";
                    std::cin  >> Sizeq;
                std::cout << "Sizeq = " << Sizeq << "\n";
                    break;

        }

        std::cout      << "\n\tFactor 1 =   " << Factor1 ;
        std::cout      << "\n\tFactor 2 =   " << Factor2 ;
        std::cout      << "\n\tRate of load increase =   " << gamma2 ;
        std::cout      << "\n\tRate of upgrades =   " << mu ;
        std::cout      << "\n\tMaximum load =   " << MaxLoadN << "\n" ;
        std::cout      << "\n\tFailing load =   " << FailLoadN << "\n" ;
        std::cout      << "\n\tMean load =   " << MeanLoad << "\n" ;
        std::cout      << "\n\tEvaluation time =   " <<Jevaluation << "\n" ;
        std::cout      << "\n\tImprovement kappa =   " << kappa << "\n" ;

        if (Iterations < 100001) kmax = 2;
        else kmax      = Iterations/50000;
        SizeDim = 2*Iterations/Jevaluation;

        LastFail       = new long[Size];             // Time since last failure
        Component      = new long[Size];             // Status of a component
        Time           = new long[Size];             // Time since being replaced
        TimeM          = new long[Size];             // Time since maintained
        Age            = new double[Size];    // Effective age of component
        Load           = new double[Size];    // Load of a component
        MaxLoad = new double[Size];     // Maximum load of component
        FailLoad       = new double[Size];     // Failing load of component
        MP             = new double[SizeDim]; // Monthly averaged daily
                                              // profit for agent 1
        TRA            = new double[SizeDim]; // Repair time
        MP2            = new double[SizeDim]; // Monthly averaged daily
                                              // profit for agent 2
        MU             = new double[SizeDim]; // Upgrate amount
        MaxP2          = new double[SizeDim];
//      MP3            = new double[SizeDim]; // Monthly averaged daily
                                              // profit for agent 3
        TRM            = new double[SizeDim];  // Maintenance time
        Maxmu          = new double[SizeDim];
        Maxday  = new double[SizeDim];
        MeanOrder      = new double[SizeDim];
        MeanMaxP2      = new double[SizeDim];
        MaxP           = new double[SizeDim];
        MaxTRA  = new double[SizeDim];
        MaxdayT = new double[SizeDim];
        MeanMaxP       = new double[SizeDim];
        MeanMaxP3      = new double[SizeDim];
        MaxP3          = new double[SizeDim];
        MaxTRM  = new double[SizeDim];
        MaxdayTm       = new double[SizeDim];

        XTauf2  = Tauf*Tauf;
        FailR          = exp(-0.5/XTauf2);
        FailR_2 = pow(FailR,4);
        FailR_3 = pow(FailR,9);
        meanF          = 0.;          // Averaged number of daily failures
        meanR          = 0.;          // Average number of components replaced
        meanMu  = 0.;
        meanTauR       = 0.;
        meanTauM       = 0.;
        CountCas       = 0.;
        CountLarge     = 0.0;
        meanD          = 0.0;
        qmid           = (1. +q0)/2.;
        std::cout      << "\n\tExponent q = ";
        std::cout      << q << "\n";
```

```
        // Preparation of output files
        fout1 <<"time"<<"\t"<<"Failures trigger"<<"\t"<<"Total failures"
                 <<"\t"<< "Duration" <<"\t"<<"Replacements"<<"\n";
        fout2 <<"time"<<"\t"<<"Size"<<"\t"<<"Duration"<<"\n";
        fout3 <<"time"<<"\t"<<" mean Load"<<"\t"<<
               " Total Load/N"<<"\t"<<" Maximum Load"<<"\t" <<
               "mean Replacements" <<"\t"<<"mean Utility 1"<<"\t"<<
               "mean Utility 2"<<"\t"<<"mean Utility 3"<<"\t"<<
               "mean Failures"<<"\t"<<"mean Cost"<<"\t"<<
               " Utilization"<<"\n";
        fout  <<"Component"<<"\t"<<"Waiting Time"<<"\n";
        fout4 <<"component" <<"\t"<< "Age" <<"\t"<<"Time"<<"\t"<<
               "Maximum load" << "\n";
        fout5 <<"Day" <<"\t"<< "Profit" <<"\t"<<"Replacement Time"<<
               "\t"<<"Decision" <<"\t"<<"q" <<"\t"<<"Max T"<<
               "\t"<<"order Max T" <<"\t"<<"Mean Max P"  <<"\t"<<
               "Mean Cost"  << "\n";
        fout6 <<"Day" <<"\t"<< "Profit" <<"\t"<<"mu - 1" <<
               "\t"<<"Decision" <<"\t"<<"q" <<"\t"<<"Max mu" <<
               "\t"<<"order Max mu" <<"\t"<<"Mean Max P"  <<"\t"<<
               "Mean Order" <<"\t"<<"Mean Cost"  << "\n";
        fout7 << "j" <<"\t"<< "Size" <<"\t"<<"Duration" << "\n";
        fout8 <<"time"<<"\t"<<"Size"<<"\t"<<"Duration"<<"\n";
        fout9 <<"time"<<"\t"<<"Size"<<"\t"<<"Duration"<<"\n";
        fout10 <<"TauR"<<"\t"<<"mu-1"<<"\t"<<"TauM"<<"\t"<<"Utility 1" <<
                "\t"<< "Utility 2" <<"\t"<< "Utility 3"<<"\t"<<
                "<Size>"<<"\t"<< "Cost"<<"\t"<<"<Duration>"<<"\t"<<
                "Number"<<"\t"<<"S>500"<<"\t"<<"Number q low"<<"\t"<<
                "S>500 q low"<<"\t"<<"Number q high"<<"\t"<<
                "S>500 q high"<<"\n";
        fout11 <<"Day" <<"\t"<< "Profit" <<"\t"<<"Maintenance Time" <<
                "\t"<<"Decision" <<"\t"<<"q" <<"\t"<<"Max T" <<"\t"<<
                "order Max T\tMean Max P\tMean Cost\n";

        // Initialization of the system
        for(int i=0; i<Size; i++)
        {
                Component[i] = 1;             // All componets OK
                LastFail[i]     = 0;          // Last time i-component failed
                Time[i]  = TauR*rand()/g;     // Random distribution of
                                            // replacement times
                TimeM[i]        = Time[i];
                Age[i]   = Time[i];           // Age of component time
                                            // since replacement
                MaxLoad[i]     = MaxLoadN;
                FailLoad[i]    = FailLoadN;
        }
        for(int i=0; i<SizeDim; i++)
        {
                MP[i]          = 0.0;
                MP2[i]  = 0.0;
                TRA[i]  = 0.0;
                MU[i]          = 0.0;
                Maxmu[i]       = 0.0;
                MaxP2[i]       = 0.0;
                Maxday[i]      = 0.0;
                MeanMaxP2[i] = 0.0;
                MeanOrder[i] = 0.0;
        }
}
```

```
void timeAdvance()
{
        // Time advanced
        long Tfails       = 0; // Total # components failing at time j
        long Treplace  = 0;    // Total # components replaced at time j
        long TMaintenance      = 0;    // Total # components serviced at time j
        long Cascade   = 0;    // Cascade size at time j
        MeanLoad               = gamma2*MeanLoad;
        p                      = p0*LoadRange*MeanLoad/Ktrans;
        double TotalLoad       = 0.;

        //Testing the components for failure
        for(int i=0; i< Size ; i++)
        {
                double rf      = rand()/g;
                double xt      = Age[i];
        //      double xt2     = xt*xt;
        //      double test = pow(FailR, xt2);
       // program optimization, limits use of expensive pow function
             double test;
             if(xt == 0.0)
                {  test = 1.0; }
                else if(xt == 1.0)
                {  test = FailR; }
                else if(xt == 2.0)
                {  test = FailR_2; }
                else if(xt == 3.0)
                {  test = FailR_3; }
                else
                {  test = pow(FailR,(xt*xt));
                }

                Load[i]     = MeanLoad*(1+2.*(rand()/g-0.5)*LoadRange);
                TotalLoad      = TotalLoad + Load[i];

                if (rf > test || Load[i]>MaxLoad[i]) // Component i fails
                {
                        Component[i]   = 0;
                        WaitTime               = j-LastFail[i];
                        LastFail[i]            = j;
                        fout.precision(10);
                        fout << i << "\t" << WaitTime << "\n";
                        fout.clear();
                        Tfails         += 1;
                }
                Time[i] += 1;
                TimeM[i]       += 1;
                Age[i]  += 1;
        }
        long TFkeep    = Tfails;      // Total failures at beginning of day j

        // Cascading event
        if (Tfails > 0)  // There is a cascading event and cascade starts
        {
                int k   = 0;           // Cascate iteration index
                Cascade = Tfails;
                while (Tfails > 0 )
                {
                        for(int i=0; i< Ktrans ; i++)
                   { // Transfer the power failed to random components
                                double trans   = rand()/g;
                                int ik         = Size*trans;
                                Age[ik]        += Tfails*Factor1;
                     // Transfer power translates in stress on components
                                Load[ik]       += Tfails*p;
                     // Transfer power increases load
                             }

                             Tfails  = 0;
```

```
                for(int i=0; i< Size ; i++)
           {  // Components can fail by increase stress
                        if (Load[i]>MaxLoad[i] && Component[i] != 0)
                        {
                                Component[i]   = 0;
                                Tfails         += 1;
                        }
                        MaxLoad[i]     = MaxLoadN-Factor2*Age[i];
                }
                Cascade += Tfails; // # compnts failed in iteration k
                                    // added to the total cascade size
                k++;
        }

        Duration       = k;

        if (j > 3*jthr)
        {
                fout2.precision(10);
                fout2 << j <<"\t"<< Cascade <<"\t"<< k <<"\n";
                fout2.clear();
                if (q < qmid)
                {
                        fout8 << j <<"\t"<< Cascade <<"\t"<< k <<"\n";
                        if (Cascade > Size/2) NCascade1= NCascade1 + 1;
                        NCascade1T     =NCascade1T +1;
                }
                if (q > qmid)
                {
                        fout9 << j <<"\t"<< Cascade <<"\t"<< k <<"\n";
                        if (Cascade > Size/2) NCascade2= NCascade2 + 1;
                        NCascade2T     =NCascade2T +1;
                }
        }
}

// Replace/Repair of old/failed components
double PiC3            = 0.0;
for(int i=0; i< Size ; i++)
{
        if (Component[i] == 0 || Time[i] > TauR-1)
        {
                Component[i]   = 1;
                Time[i]        = 0;
                Age[i]         = 0;
                Treplace       += 1;
        }
        if (Component[i] != 0 && TimeM[i] > TauM-1)
        {
                Age[i]         = kappa*Age[i];
                TMaintenance   += 1;
                PiC3           = 1.0;
                TimeM[i]       = 0;
        }
}

if (Cascade > 10)
{
        MaxLoadN       = mu*MaxLoadN;
        Factor2        = mu*Factor2;
}

if (j > Iterations - 400000) fout7 << j <<"\t"<< Cascade <<"\t"<<
                                Duration<<"\n";

if ( TFkeep > 0 || Treplace > 0)
{
        fout1.precision(10);
        fout1 << j <<"\t"<< TFkeep <<"\t"<< Cascade
```

```
                    <<"\t"<< Duration <<"\t"<< Treplace <<"\n";
        fout1.clear();
}
double Xjx     = j;
double XSize   = Size;
double XCas    = Cascade/XSize;
double PiC     = 0.0;
//double PiC2  = 0.0;
PiC2           = 0.0;
if( XCas > Sizeq/XSize) jlast = j;
switch (iq) {
        case 1:
        break;

        case 2:
        if( XCas > Sizeq/XSize)
        {
             j00               = j;
                jkk            = 0;
        }
        q       = 1.;
        if (Xjx-j00 < 5*Tauq) q       = 1 + (q0-1)*exp(-(Xjx-j00)/Tauq);
        break;

        case 3:
        if( j - jlast > Tauq)
        {
                q       = q0;
                double XZZ     = Xjx-jlast;
                if (XZZ < 10*TauD) q   = 1 + (q0-1)*tanh(XZZ/TauD);
        }
        else
        {
                q       = 1;
        }
        break;
}

PiC     = 0.0;
PiC2    = 0.0;
if ( XCas > 0)
{
        PiC     = pow(XCas, q)/(pow(XCas, q)+pow(1-XCas, q));
}

if ( Cascade > 10) PiC2       = 1.0;

Profit  = 2.*(Size-Cascade)-(Treplace+100*XSize*PiC);
Profit2 = 2.*(Size-Cascade)-(10000*2000*(mu-1)*
            (mu-1)*PiC2+1000.*(mu-1.)+100*XSize*PiC);
Profit3 = 2.*(Size-Cascade)-(0.3*TMaintenance+100*XSize*PiC);
Cost           = Treplace + 0.3*TMaintenance +10000*2000*(mu-1)*
            (mu-1)*PiC2+1000.*(mu-1.);

if (j > 3*jthr)
{
        XXj             = XXj + 1;
        meanR           = (Treplace+XXj*meanR)/(XXj+1.0);
        meanM           = (TMaintenance+XXj*meanM)/(XXj+1.0);
        meanMu  = (mu+XXj*meanMu)/(XXj+1.0);
        meanTauR        = (TauR+XXj*meanTauR)/(XXj+1.0);
        meanTauM        = (TauM+XXj*meanTauM)/(XXj+1.0);
        meanP           = (Profit+XXj*meanP)/(XXj+1.0);
        meanP2  = (Profit2+XXj*meanP2)/(XXj+1.0);
        meanP3  = (Profit3+XXj*meanP3)/(XXj+1.0);
        meanCostT       = (Cost+XXj*meanCostT)/(XXj+1.0);
        if (XCas > 0.5) CountLarge = CountLarge +1;
}
if (XCas > 0 && j > 3*jthr )
```

```
        {
                XXk             = XXk+1;
                meanF           = (Cascade+XXk*meanF)/(XXk+1.0);
                meanD           = (Duration+XXk*meanD)/(XXk+1.0);
                CountCas        = CountCas + 1;
        }
        Duration        = 0;

        if (kc == kmax)
        {
                Utilization     =TotalLoad/(XSize*MaxLoadN);
                fout3.precision(10);
                fout3   << j+1 <<"\t"<< MeanLoad <<"\t"<< TotalLoad/XSize <<
                   "\t"<< MaxLoadN <<"\t"<<meanR<<"\t"<< meanP <<"\t"<<
                   meanP2 <<"\t"<< meanP3 <<"\t"<< meanF<<"\t"<<
                   meanCostT <<"\t"<< Utilization <<"\n";
                fout3.clear();
                kc      = 0;
        }
} // end timeAdvance

void beforeAgents(int jk)
{
        MP[jk]  = MeanProfit/XJ;
        TRA[jk] = TauR;
        MP2[jk] = MeanProfit2/XJ;
        MU[jk]  = mu - 1.0;
        MP3[jk] = MeanProfit3/XJ;
        TRM[jk] = TauM;
        Xprofit = MeanProfit/XJ;
        meanCost        = meanCost/XJ;
}

void afterAgents(int jk)
{
        MeanProfit      = 0.0;
        MeanProfit2     = 0.0;
        MeanProfit3     = 0.0;
        XPiC2           = 0.0;
        je              = 0;
        fout5.precision(10);
        fout5 << j <<"\t"<< MP[jk] <<"\t"<<TRA[jk] <<"\t"<< Ak <<"\t"<< q
            <<"\t"<< XXMMT <<"\t"<< ijt<<"\t"<< MeanMaxT <<"\t"<<
            meanCost<< "\n";
        fout5.clear();
        fout6.precision(10);
        fout6 << j <<"\t"<< MP2[jk] <<"\t"<<MU[jk] <<"\t"<< Ak2 <<"\t"<<q
            <<"\t"<< XXmm <<"\t"<< ijk<<"\t"<< MeanMax <<"\t"<<
            MeanOrder[ijk] <<"\t"<< meanCost<< "\n";
        fout6.clear();
        fout11.precision(10);
        fout11 << j <<"\t"<< MP3[jk] <<"\t"<<TRM[jk] <<"\t"<< Ak3 <<
             "\t"<< q <<"\t"<< XXMMT3 <<"\t"<< ijt3<<"\t"<< MeanMaxTm
             <<"\t"<< meanCost<< "\n";
        fout11.clear();
}


void updateProfits()
{
        MeanProfit      = MeanProfit + Profit;
        MeanProfit2     = MeanProfit2 + Profit2;
        MeanProfit3     = MeanProfit3 + Profit3;
        meanCost        = meanCost + Cost;
        XPiC2           = XPiC2 + PiC2;
}
int closeFiles()
{
        std::cout << "Failures per unit time =  " << meanF << "\n";
```

```
        std::cout << "Replacements per unit time =  " << meanR << "\n";
        std::cout << "Maintenance per unit time =  " << meanM << "\n";
        std::cout << "Utility 1 =  " << meanP << "\n";
        std::cout << "Utility 2 =  " << meanP2 << "\n";
        std::cout << "Utility 3 =  " << meanP3 << "\n";
        std::cout << "True Cost =  " << meanCostT << "\n";
        std::cout << "Replacement Time =  "<< meanTauR<<"\n";
        std::cout << "Maintenance Time =  "<< meanTauM<<"\n";
        std::cout << "Upgrade rate mu =  "<< meanMu<<"\n";
        fout4.precision(10);
        for(int i=0; i< Size ; i++) fout4 << i <<"\t"<< Age[i] <<"\t"<<
                                    Time[i]<<"\t"<< MaxLoad[i]<< "\n";
        fout4.clear();

        fout10 <<meanTauR<<"\t"<< meanMu-1<<"\t"<< meanTauM<<"\t"<<
             meanP <<"\t"<< meanP2 <<"\t"<< meanP3 <<"\t"<< meanF <<
             "\t"<<meanCostT<<"\t"<<meanD<<"\t"<<CountCas<<"\t"<<
             CountLarge<<"\t"<<NCascade1T<<"\t"<<NCascade1<<"\t"<<
             NCascade2T<<"\t"<<NCascade2<<"\n";

        fout.close();
        fout1.close();
        fout2.close();
        fout3.close();
        fout4.close();
        fout5.close();
        fout6.close();
        fout7.close();
        fout8.close();
        fout9.close();
        fout10.close();
        fout11.close();

        stop_time = clock();
        std::cout << "Time taken = " << (stop_time - start_time) /
                   (CLOCKS_PER_SEC * 60.) << " min. \n";

        std::cout << "Program ends";
      return 0;
}

#ifdef REPAST

/* Functions used by Agent1 */
double getMaxPValue(int index) { return MaxP[index]; }
void setMaxPValue(int index, double val) { MaxP[index] = val; }
double getMeanMaxPValue(int index) { return MeanMaxP[index]; }
void setMeanMaxPValue(int index, double val) { MeanMaxP[index] = val; }
double getMaxTRAValue(int index) { return MaxTRA[index]; }
void setMaxTRAValue(int index, double val) { MaxTRA[index] = val; }
double getMaxdayTValue(int index) { return MaxdayT[index]; }
void setMaxdayTValue(int index, double val) { MaxdayT[index] = val; }
double getMPValue(int index) { return MP[index]; }
void setMPValue(int index, double val) { MP[index] = val; }
double getTRAValue(int index) { return TRA[index]; }
void setTRAValue(int index, double val) { TRA[index] = val; }

/* Functions used by Agent2 */
double getMaxP2Value(int index) { return MaxP2[index]; }
void setMaxP2Value(int index, double val) { MaxP2[index] = val; }
double getMeanMaxP2Value(int index) { return MeanMaxP2[index]; }
void setMeanMaxP2Value(int index,double val){ MeanMaxP2[index] = val; }
double getMaxmuValue(int index) { return Maxmu[index]; }
void setMaxmuValue(int index, double val) { Maxmu[index] = val; }
double getMaxdayValue(int index) { return Maxday[index]; }
void setMaxdayValue(int index, double val) { Maxday[index] = val; }
double getMP2Value(int index) { return MP2[index]; }
void setMP2Value(int index, double val) { MP2[index] = val; }
double getMUValue(int index) { return MU[index]; }
```

```
void setMUValue(int index, double val) { MU[index] = val; }

/* Functions used by Agent3 */
double getMaxP3Value(int index) { return MaxP3[index]; }
void setMaxP3Value(int index, double val) { MaxP3[index] = val; }
double getMeanMaxP3Value(int index) { return MeanMaxP3[index]; }
void setMeanMaxP3Value(int index,double val){ MeanMaxP3[index] = val; }
double getMaxTRMValue(int index) { return MaxTRM[index]; }
void setMaxTRMValue(int index, double val) { MaxTRM[index] = val; }
double getMaxdayTmValue(int index) { return MaxdayTm[index]; }
void setMaxdayTmValue(int index, double val) { MaxdayTm[index] = val; }
double getMP3Value(int index) { return MP3[index]; }
void setMP3Value(int index, double val) { MP3[index] = val; }
double getTRMValue(int index) { return TRM[index]; }
void setTRMValue(int index, double val) { TRM[index] = val; }

#else

// Agent functions

double Agent1(int Jevaluation,double *MaxP,double *MeanMaxP, double *MaxTRA, double
*MaxdayT, double *MP,double *TRA, double *pt_MeanMaxT,double *pt_XXMMT, int *pt_Ak,int j,
int jk, int jkk, int *pt_kkj, int *pt_ijt, double TauR)
{
        int     kkkj    = *pt_kkj;
        int Nexplor     = 22;
        int Ntrans      = Nexplor*Jevaluation;
        int kijt        = *pt_ijt;
        double MeanMax1         = *pt_MeanMaxT;
        if (jk <= 1)
        {
                kijt    = 1;
                MeanMax1 = 0.;
        }

        if (MP[jk] > MeanMax1 && j < Ntrans + 1)
        {
                kijt            = kijt + 1;
                MaxP[kijt]  = MP[jk];
                MaxTRA[kijt] = TRA[jk];
                MaxdayT[kijt] = jk;
                MeanMaxP[kijt-1] = MeanMax1;
                MeanMax1    = 0.0;
        }
        if (MP[jk] > 1.0*MeanMax1 && j > Ntrans)
        {
                kijt          = kijt + 1;
                MaxP[kijt]      = MP[jk];
                MaxTRA[kijt] = TRA[jk];
                MaxdayT[kijt] = jk;
                MeanMaxP [kijt-1] = MeanMax1;
                MeanMax1        = 0.0;
        }
        double XX       = jk - MaxdayT[kijt];
        MeanMax1        = (MP[jk]+XX*MeanMax1)/(XX+1.0);

        if (MeanMax1 < 0.98*MeanMaxP[kijt-1])
        {
                MeanMaxP [kijt]         = MeanMax1;
                int jmax                = 1;
                double XX               = 0.;
                double XY               = 0.;
                for (int i = 2 ; i<kijt + 1; i++)
                {
                        if (MeanMaxP[i]>MeanMaxP[i-1] ) jmax = i;
                        XX      = XX + MeanMaxP[i]*MaxTRA[i];
                        XY      = XY + MeanMaxP[i];
                }
                if (jmax == 1) jmax = kijt;
```

```
        kijt                    = kijt + 1;
        if ( rand()/g > 0.25)
        {
               MaxP[kijt]     = MeanMaxP[jmax];
               MaxTRA[kijt] = MaxTRA[jmax];
               MaxdayT[kijt] = jk;
        }
        else
        {
               MaxP[kijt]     = XY/(kijt-2);
               MaxTRA[kijt] = 2.;
               if (XY > 0) MaxTRA[kijt] = XX/XY;
               MaxdayT[kijt] = jk;
        }
        MeanMax1               = MeanMaxP[jmax];

    }

    double MMT     = MaxTRA[kijt];
    if (MMT < 1) MMT       = 2.;
    if (jk < 31 || jkk < 3) TauR  = TauR - 6;
    if (jk > 30 && jkk > 2)
    {
        double A       = 1.0;
        double B       = 0.0;
        if( fabs(TRA[jk-1]-TRA[jk-2]) > 0.0001 &&
         fabs(TRA[jk-1]-TRA[jk-3]) > 0.0001 &&
         fabs(TRA[jk-2]-TRA[jk-3])>0.0001)
        {
               A       = ((MP[jk-1]-MP[jk-2])/(TRA[jk-1]-TRA[jk-2])-
                    (MP[jk-1]-MP[jk-3])/(TRA[jk-1]-TRA[jk-3]))/
                    (TRA[jk-2]-TRA[jk-3]);
               B       = (MP[jk-1]-MP[jk-2])/(TRA[jk-1]-TRA[jk-2])-
                    A*(TRA[jk-1]+TRA[jk-2]);
        }

        //std::cout << A << "\t" << B <<   "\n";

        if (A < 0)
        {     TauR             = -B/(2.*A);
               *pt_Ak  = 1;
        }
        else
        {     *pt_Ak   = 2;
               TauR           = MMT;

               if ( rand()/g > 0.90)
               {
                      TauR    = MMT + 10.*(rand()/g-0.5);
                      kkkj++;
                      *pt_Ak = 3;
               }
        }
        if (TauR < 0) TauR     = 2.;
        if (TauR > 10.*MMT) TauR = MMT + 10.*(rand()/g-0.5);
    }

    *pt_kkj = kkkj;
    if (TauR > 500) TauR = 500.;
    if (TauR < 1) TauR   = 2.;
    *pt_XXMMT              = MMT;
    *pt_MeanMaxT   = MeanMax1;
    *pt_ijt        = kijt;

    return TauR;
}
```

```
double Agent2(int Jevaluation, double *MaxP2, double *MeanMaxP2, double *Maxmu, double
*Maxday,double *MP2,double *MU, double *pt_MeanMax, double *pt_XXmm, int *pt_Ak2,int j,
int jk, int jkk, int *pt_ijk, double *pt_gamma, double mu)
{
        int Nexplor    = 22;
        int Ntrans     = Nexplor*Jevaluation;
        int kijk       = *pt_ijk;
        double Xmu     = mu - 1.0;
        double Xg      = *pt_gamma;
        double MeanMax1 = *pt_MeanMax;
        if (jk <= 1)
        {     kijk     = 1;
                MeanMax1       = 0.;
        }
        if (MP2[jk] > MeanMax1 && j < Ntrans + 1)
        {
                kijk                          = kijk + 1;
                MaxP2[kijk]                   = MP2[jk];
                Maxmu[kijk]                   = MU[jk];
                Maxday[kijk]          = jk;
                MeanMaxP2[kijk-1]           = MeanMax1;
                MeanMax1                      = 0.0;
        }
        if (MP2[jk] > 1.05*MeanMax1 && j > Ntrans)
        {
                kijk                          = kijk + 1;
                MaxP2[kijk]                   = MP2[jk];
                Maxmu[kijk]                   = MU[jk];
                Maxday[kijk]          = jk;
                MeanMaxP2[kijk-1]           = MeanMax1;
                MeanMax1                      = 0.0;
        }
        double XX      = jk - Maxday[kijk];
        MeanMax1       = (MP2[jk]+XX*MeanMax1)/(XX+1.0);

        if (MeanMax1 < 0.95*MeanMaxP2[kijk-1])
        {
                MeanMaxP2[kijk]       = MeanMax1;
                int jmax              = 1;
                double XX             = 0.;
                double XY             = 0.;
                for (int i = 2 ; i<kijk + 1; i++)
                {
                        if (MeanMaxP2[i]>MeanMaxP2[i-1] ) jmax       = i;
                        XX      = XX + MeanMaxP2[i]*Maxmu[i];
                        XY      = XY + MeanMaxP2[i];
                }
                if (jmax == 1) jmax = kijk;
                kijk                    = kijk + 1;
                if ( rand()/g > 0.25)
                {
                        MaxP2[kijk]           = MeanMaxP2[jmax];
                        Maxmu[kijk]           = Maxmu[jmax];
                        Maxday[kijk]   = jk;
                }
                else
                {
                        MaxP2[kijk]           = XY/(kijk-2);
                        Maxmu[kijk]           = Maxmu[jmax];
                        if (XY > 0) Maxmu[kijk]       = XX/XY;
                        Maxday[kijk]   = jk;
                }
                MeanMax1              = MeanMaxP2[jmax];
        }

        double MMU     = Maxmu[kijk];
        if (MMU < Xg-1) MMU    = 3.*(Xg-1);
        if (jk < Nexplor + 1 || jkk < 3) Xmu = Xmu*1.4;
        if (jk > Nexplor && jkk > 2)
```

```
        {
                double A       = 1.0;
                double B       = 0.0;
                if( fabs(MU[jk-1]-MU[jk-2]) > 0.000001 &&
                 fabs(MU[jk-1]-MU[jk-3]) > 0.000001 &&
                 fabs(MU[jk-3]-MU[jk-2])>0.000001)
                {
                        A      = ((MP2[jk-1]-MP2[jk-2])/(MU[jk-1]-MU[jk-2])-
                             (MP2[jk-1]-MP2[jk-3])/(MU[jk-1]-MU[jk-3]))/
                             (MU[jk-2]-MU[jk-3]);
                        B      = (MP2[jk-1]-MP2[jk-2])/(MU[jk-1]-MU[jk-2])-
                             A*(MU[jk-1]+MU[jk-2]);
                }
                if (A < 0)
                {     Xmu              = -B/(2.*A);
                        *pt_Ak2 = 1;
                }
                else
                {     *pt_Ak2  = 2;
                        Xmu            = MMU;
                        if ( rand()/g > 0.95)
                        {
                                Xmu     = MMU*(1.0 + 2.*(rand()/g-0.5));
                                *pt_Ak2 = 3;
                        }
                }
        }
        if (Xmu <  Xg-1) Xmu = 3.*(Xg-1);
        if (Xmu > 1.) Xmu      = 1.0;

        *pt_XXmm       = MMU;
        *pt_ijk = kijk;
        *pt_MeanMax    = MeanMax1;
        //std::cout << MeanMax1 << "\t" << kijk <<   "\n";

        return 1.+Xmu;
}
double Agent3(int Jevaluation,double *MaxP3,double *MeanMaxP3, double *MaxTRM, double
*MaxdayTm, double *MP3,double *TRM, double *pt_MeanMaxTm,double *pt_XXMMT3, int
*pt_Ak3,int j, int jk, int jkk, int *pt_kkj3, int *pt_ijt3, double TauM)
{
        int     kkkj    = *pt_kkj3;
        int Nexplor    = 22;
        int Ntrans     = Nexplor*Jevaluation;
        int kijt       = *pt_ijt3;
        double MeanMax1 = *pt_MeanMaxTm;
        if (jk <= 1)
        {
                kijt    = 1;
                MeanMax1 = 0.;
        }
        if (MP3[jk] > MeanMax1 && j < Ntrans + 1)
        {
                kijt                          = kijt + 1;
                MaxP3[kijt]                   = MP3[jk];
                MaxTRM[kijt]          = TRM[jk];
                MaxdayTm[kijt]        = jk;
                MeanMaxP3[kijt-1]           = MeanMax1;
                MeanMax1                      = 0.0;
        }
        if (MP3[jk] > 1.0*MeanMax1 && j > Ntrans)
        {
                kijt                          = kijt + 1;
                MaxP3[kijt]                   = MP3[jk];
                MaxTRM[kijt]          = TRM[jk];
                MaxdayTm[kijt]        = jk;
                MeanMaxP3 [kijt-1]    = MeanMax1;
                MeanMax1                      = 0.0;
        }
```

```
double XX      = jk - MaxdayTm[kijt];
MeanMax1       = (MP3[jk]+XX*MeanMax1)/(XX+1.0);

if (MeanMax1 < 0.98*MeanMaxP3[kijt-1])
{
       MeanMaxP3[kijt]       = MeanMax1;
       int jmax              = 1;
       double XX             = 0.;
       double XY             = 0.;
       for (int i = 2 ; i<kijt + 1; i++)
       {
              if (MeanMaxP3[i]>MeanMaxP3[i-1] ) jmax       = i;
              XX      = XX + MeanMaxP3[i]*MaxTRM[i];
              XY      = XY + MeanMaxP3[i];
       }
       if (jmax == 1) jmax = kijt;
       kijt                  = kijt + 1;
       if ( rand()/g > 0.25)
       {
              MaxP3[kijt]           = MeanMaxP3[jmax];
              MaxTRM[kijt]   = MaxTRM[jmax];
              MaxdayTm[kijt] = jk;
       }
       else
       {
              MaxP3[kijt]           = XY/(kijt-2);
              MaxTRM[kijt]   = 2.;
              if (XY > 0) MaxTRM[kijt] = XX/XY;
              MaxdayTm[kijt] = jk;
       }
       MeanMax1              = MeanMaxP3[jmax];
}

double MMT     = MaxTRM[kijt];
if (MMT < 1) MMT       = 2.;
if (jk < 31 || jkk < 3) TauM  = TauM - 4;
if (jk > 30 && jkk > 2)
{
       double A       = 1.0;
       double B       = 0.0;
       if( fabs(TRM[jk-1]-TRM[jk-2]) > 0.0001 &&
        fabs(TRM[jk-1]-TRM[jk-3]) > 0.0001 &&
        fabs(TRM[jk-2]-TRM[jk-3])>0.0001)
       {
              A = ((MP3[jk-1]-MP3[jk-2])/(TRM[jk-1]-TRM[jk-2])-
               (MP3[jk-1]-MP3[jk-3])/(TRM[jk-1]-TRM[jk-3]))/
               (TRM[jk-2]-TRM[jk-3]);
              B = (MP3[jk-1]-MP3[jk-2])/(TRM[jk-1]-TRM[jk-2])-
               A*(TRM[jk-1]+TRM[jk-2]);
       }

       //std::cout << A << "\t" << B <<   "\n";

       if (A < 0)
       {     TauM            = -B/(2.*A);
              *pt_Ak3 = 1;
       }
       else
       {     *pt_Ak3  = 2;
              TauM           = MMT;

              if ( rand()/g > 0.90)
              {
                     TauM    = MMT + 10.*(rand()/g-0.5);
                     kkkj++;
                     *pt_Ak3 = 3;
              }
       }
```

```
                if (TauM < 0) TauM     = 2.;
                if (TauM > 10.*MMT) TauM = MMT + 10.*(rand()/g-0.5);
        }
        *pt_kkj3        = kkkj;
        if (TauM > 500) TauM   = 500.;
        if (TauM < 1) TauM     = 1.;
        *pt_XXMMT3             = MMT;
        *pt_MeanMaxTm  = MeanMax1;
        *pt_ijt3               = kijt;

        return TauM;
}
#endif
```

Appendix 2: Tau Profiling Tool Makefile

```
#***************************************************************
#*                    TAU Portable Profiling Package
**
#*                    http://www.cs.uoregon.edu/research/tau
**
#***************************************************************

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
#
#    Adapted for Midnight SAMPLES_HOME by E. Kornkven, ARSC.
#
#    This is the Makefile for compiling the example program using TAU.
#    To change experiments, change the value of TAUMAKEFILE.
#    See TAUMAKEFILEDIR for all experiment Makefiles.
#
#    To disable TAU in a program that has TAU_START() and TAU_STOP()
#    calls in it, change the Makefile as follows (by uncommenting the
#    appropriate line below):
#    1) Change CC back to mpicc;
#    2) Define TAU_DISABLE_API to the preprocessor which will convert
#       TAU_START() and TAU_STOP() calls to null strings;
#    2) Specify the path (using the -I compiler option) to TAU.h to
#       CFLAGS.
#       CFLAGS is already set that way below.
#
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

TARGET  = cascadeModel

#TAUMAKEFILE   = Makefile.tau-pathcc-mpi-pdt
TAUMAKEFILE = Makefile.tau-pathcc-pdt
#TAUMAKEFILE   = Makefile.tau-pathCC-mpi-pdt
# For an example of a different experiment, uncomment the following
# line.
#TAUMAKEFILE   = Makefile.tau-callpath-pathcc-mpi-pdt
TAUROOTDIR=$(PET_HOME)/tau
TAUMAKEFILEDIR=$(TAUROOTDIR)/x86_64/lib
TAUINCLUDEDIR=$(TAUROOTDIR)/include
TAUMAKEFLAGS   = -tau_makefile=$(TAUMAKEFILEDIR)/$(TAUMAKEFILE) -optKeepFiles

CC              = $(TAUROOTDIR)/x86_64/bin/tau_cc.sh $(TAUMAKEFLAGS)
# Uncomment the following line to compile with TAU disabled.
#CC             = mpicc -DTAU_DISABLE_API
CFLAGS  = -I$(TAUINCLUDEDIR)
LDFLAGS =
CXX             = $(TAUROOTDIR)/x86_64/bin/tau_cxx.sh $(TAUMAKEFLAGS)
F90             = $(TAUROOTDIR)/x86_64/bin/tau_f90.sh $(TAUMAKEFLAGS)

##########################################

all:            $(TARGET)

$(TARGET):      $(TARGET).o
        $(CXX) $(LDFLAGS) $< -o $@ -lstdc++

$(TARGET).o : $(TARGET).cpp
        $(CXX) $(CFLAGS) -c $<

clean:
        $(RM) $(TARGET).o $(TARGET).d $(TARGET)

##########################################
```

Appendix 3: OPA Model Source Code: "opaModel.cpp"

```
/*****************************************************************
** Fllename:  opaModel.cpp
** Date:      March 2009
** Notes:     This version of OPA has been modularized to ease
             debugging and profiling efforts. ONN Nov/Dec 2008
             REPAST compatible  ONN Mar 2009


//OPA code documented in the following papers:
//
//I. Dobson, B.A. Carreras, V. Lynch, D.E. Newman, "An initial model //for complex
dynamics in electric power system blackouts,"
//34th Hawaii International Conference on System Sciences, Maui, //Hawaii, January 2001.
Available from IEEE at //http://ieeexplore.ieee.org.
//
//B.A. Carreras, V.E. Lynch, M. L. Sachtjen, I. Dobson, D.E. Newman, //Modeling blackout
dynamics in power transmission networks
//with simple structure," 34th Hawaii International Conference on //System Sciences,
Maui, Hawaii, January 2001.
// Available from IEEE at http://ieeexplore.ieee.org.
//
//B.A. Carreras, V.E. Lynch, I. Dobson, and D.E. Newman, CHAOS, 14, 643 (2004).

*****************************************************************/

#define EPS 1.0e-6
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include <time.h>      // for clock(), CLOCKS_PER_SEC
#include <string.h>
#include <ctype.h>
using namespace std;
/*****************************
       Globals
*****************************/
double lmult;
int ieee=1;
ofstream fout5("Averaged");
ofstream fout7("Time_var_blackout");
ofstream fout2("iterations_outages");
ofstream fout4("iterations_powershed");

/*****************************
       Constants
*****************************/
#define NumSize                        8
#define WinH                   400
#define WinW                   400
#define WinAx                          4*NumSize


/*****************************
       Nodes Class
*****************************/
class Node {
       public:
        int locX, locY;         // Position of the node in the grid
          int conN;            // Number of conection in and out of the node
          double Power;     // Power associated with a load
          double PowerAv;   // Average Power associated with the node
          double PowerLimit;// Maximum power in case of a generator
          int *xin;         // List of nodes with conections to this node
          double theta;          // Angle of Power
          double Generator; // Power in Generator
```

```
          double PowerShed; // Power demand not met
          int Type;              // 1=Load, 2=Gen, 3=Load+Gen, 4=nothing
          double Demand;         // Power demand for loads
          int PDFblackouts; // sum blackouts on this node
                            // (PDF written to Averaged)
          int zone;              // region for random number for gamma
          double Q;
          double deltaQ;
          char name;

          Node ( int x, int y, int n, double iz, double izL, double gen,
                int type, int vin [], char name[])
        {
                locX           = x;
             locY              = y;
             conN              = n;
             PowerAv    = iz;
                Power          = PowerAv;
        PowerLimit     = izL;
                Generator      = gen;
        Type           = type;
                Powershed      = 0;
        PDFblackouts= 0;
                Demand  = 0;
                xin         = new int[n];
                zone           = name[0];
                Q              = 0;
                deltaQ  = 0;
                for (int k = 0; k < n ; k++) xin[k] = vin[k];
           };
           Node (void) { conN = 0;};
};


/****************************
        Lines Class
****************************/
class PLine {
        public:
                int Node1,Node2;       // Nodes associated with this line
    double Z;          // Impedence of line
    double Zinit;          // Initial impedence of line
    double Flow;           // Power flow of line
    double Fmax;           // Power flow maximum of line
    double LFmax;          // Fmax without overloading
    double Overload;  // Flag for a line that overloads
    int PDFoutages;   // sum of outages triggering blackouts
                      //(PDF written)
    bool Out;      // Flag for line outage
    int Out1;      // Flag for outage on iteration 1
    double MFlow1;    // flow/fmax on 1st iteraton

    PLine ( int one, int two, double iz, double fmax){
       Node1                          = one;
      Node2                           = two;
      Z                       = iz;
      Fmax                            = fmax;
      LFmax                           = fmax;
      Zinit                           = iz;
      Overload                = 0;
      PDFoutages                      = 0;
      Out                             = false;
      Out1                            = 0;
      MFlow1                  = 0;
    };
    PLine (void) {Node1 = 0;Node2 = 0;};
};
```

```
/*****************************
       Functions prototypes
*****************************/
int NodeCount(ifstream&);
Node Build(ifstream&, bool&);
PLine Build(ifstream&, int, int);
int dec (int n, double **a, int *ip);
void sol (int n, double **a, double *b, int *ip);
void simplx(double **a,int m,int n,int m1,int m2,int m3,int *icase,
            int izrov[],int iposv[]);
void simp1(double **a,int mm,int ll[],int nll,int iabf,int *kp,
           double *bmax);
void simp2(double **a,int n,int l2[],int nl2,int *ip,int kp,
           double *q1);
void simp3(double **a,int i1,int k1,int ip,int kp);
int *ivector(int nl,int nh);
void free_ivector(int *v,int nl);
double **matrix(int nrl,int nrh,int ncl,int nch);
int Bhat(int BSize,int Nlines, PLine *lines,double **B,
          double **Binv,double *rhs, int *ip);
void Amatrix(int M, int N,int ArSize,int Nlines,PLine *lines,
             Node *nodes,double **a,double **Binv);
int flows(double eps,int ArSize,int Nlines,PLine *lines,
          Node *nodes,double **B,double *rhs, int *ip);
void ReStart();
void Swap(int *x,int *y);
void getInputParameters(int & ieee,int & start,int & AnaTime,
                        double & lambda, double & gamma_reg,
                        double & mu,double & p0,double & p1,
                        double & zmult,double & eps,
                        double & CritMargin,double & incrGenP,
                        int & days_gen,double & lmult,int & cc);
void buildNetwork(int *&ip,double **&a,int *&j00,double **&B,
                  double **&Binv,int& BSize,double& eps,bool& done,
                  int& ieee, double *&FractOverload,Node*& nodes,
                  int& ArSize,double *&Paveraged,PLine *&lines,
                  int& Nlines,double *&rhs);
int calculateBhat(int &ArSize,double **&B,double **&Binv,int &BSize,
                  int *&ip,double &eps,int &Nlines, PLine *&lines,
                  Node *&nodes,double *&rhs);
void exitRoutine(int);
void labelOutputFiles();
int timeLoop();
void outputSummary();
void countGenerators(int *&j00,int *&generatorNo,int& nPg,
                     double **&a,int *&izrov,int *&iposv,
                     int &ArSize,double **&B,int& BSize,
                     double **&Binv,int& M,int &N,Node *&nodes,
                     double &Ptotal0,double &deltaP,int &Nlines,
                     PLine *&lines,double *&rhs,int *&ip);
int innerTimeStepIterations();
int initializeSystem();
int finalizeSystem();
double g = RAND_MAX;                 // Largest possible random number
ifstream fin;
ifstream fin2;
char fname[80];                      // network file name
char fname2[80];                        // line file name
ofstream fout;
clock_t c,start_time,stop_time;     // used for timing program

/* global variables */
int countY = 0;
int TotalTime = 1;
int time_d = 1;
int AnaTime, ArSize, start, Nlines;
double AVblackouts, AVoutages, Tblackouts, Toutages;
double *FractOverload;
PLine *lines;
```

```
Node *nodes;
double* Paveraged;
double xran[123];
int overloadTotal;
double CritMargin, gamma_reg;
int *j00;
double **B;
int BSize, blackout, days_gen, icase, iterations;
int M, N, outages, RealTime;
double **Binv;
int *ip;
double *rhs;
double Dtotal, eps, lambda, PowerShedIter;
double PowerShedTotal, Ptotal, zmult;
double **a;
int *iposv;
int *izrov;
double *Out_iter;
double *PS_iter;
int nPg;
double deltaP, Ptotal0, SumFmax, SumFmaxm1, mu, p0, p1, incrGenP;
int *generatorNo;
int icount,icountm1;
double TotalSumFmax;


#ifdef REPAST

/* For simulating the C rand # generator */
int getNextRand();
double getRAND_MAX();

#else

int main (int argc, char * const argv[]) {
   initializeSystem();
   time_d = 1;

   int timeIters = 1;
   while (time_d <= TotalTime)
   {  timeIters = timeLoop();
   }
   time_d = timeIters;
   finalizeSystem();
}

#endif

int initializeSystem()
{
        /*****************************
               Variables
        *****************************/
        int cc;              // random number generator seeds
        AnaTime = 1;   // # of time_d steps that data is collected
      TotalTime =     1;    // # of time_d steps in run.  Set by user
        p0 = 0;        // probability of random line failure
        zmult = 1000;   // multiplier for Z and div of Fmax if line fails
        p1 = 0;       // probability of outage if line is overloaded
        lambda = 1;           /* regular growth for all generators and
                              power loads */
        mu = 1;       // probability of a failed line being improved
                         // if it resulted in a blackout
        eps = 0.1;            // epsolon: distance from Fmax before a line is
                         // considered overloaded
        gamma_reg = 1; // powerloads can change as much as 2-gamma
                         // to gamma
        icase   = 0;          // error returned from simplex solver
//      int overload = 0;     // # of new lines overloaded,
```

```
                // returned from flows
  Ptotal = 0;           // Total Power of all the loads
  Dtotal = 0;           // Total Power of all the loads
  deltaP = 0;           // closeness to maximum power
  iterations = 0;       // # of iterations in inner time_d loop
  bool done = false;// Flag, done building network
  //int count    = 0;    // # of line outages due to p0 in 1 time_d step
  blackout = 0;  // Flag, Blackout of TP/TPS > 1.e-5
  outages = 0;   // total # of line outages
  start = 0;            // Start time_d for data aquisition
  RealTime = 0;  // Time from start of data aquisition
  Tblackouts = 0;       // Total # in PDFblackouts
  Toutages = 0;  // Total # in PDFoutages
  AVblackouts = 0;      // Average # in PDFblackouts
  AVoutages = 0; // Average # in PDFoutages
  days_gen = 365;
M = 0;
N = 0;
countY = 0;
overloadTotal = 0;
getInputParameters(ieee,start,AnaTime,lambda,gamma_reg,mu,p0,p1,
                zmult,eps,CritMargin,incrGenP,days_gen,lmult,cc);

  /*****************************
  Get seed for random number generator
  *****************************/
  c       = time(NULL);
  cc = c;

  srand(cc);
  start_time = clock();

  // Count number of nodes
  ifstream ftry(fname);
  if(!ftry) {
         std::cout << "\n\ncannot open " << fname;
         ftry.close();
         return 1;
  } else {
         ArSize = NodeCount(ftry);
         ftry.close();
  }

  // New arrays and output files
  nodes = new Node[ArSize+1];                   // Networks of nodes
  lines = new PLine[(ArSize*(ArSize+1))/2];     // Networks of lines

  /*****************************
         Simplex Variables
  *****************************/
  BSize = ArSize-1;
  B = new double* [BSize];      // B matrix

  // Allocate space for B matrix
  for (int i = 0; i < BSize; i++)
  B[i] = new double[BSize];
  rhs = new double[BSize];
  ip = new int[BSize];          // right hand side array

  // pivot information vector obtained from dec
  Binv = new double* [BSize];   // B inverse matrix
  for (int i = 0; i < BSize; i++)
  Binv[i] = new double[BSize];

  /*****************************
         Diagnostic Variables
  *****************************/
// Fraction of overloading for each line
  FractOverload  = new double [(ArSize*(ArSize+1))/2];
```

```
        Paveraged = new double [ArSize]; // Average power for each line
      PS_iter = new double [11];
        Out_iter = new double [11];

        double** Image;
        Image = new double* [600];
        for (int i = 0; i < 600; i++)
        Image[i] = new double[600];

      // Open files for output
        char name[20]="ReStart0";
        if(fname[0] == 'R') name[7]=fname[strlen(fname)-1]+1;
        std::cout <<"Restart file in OPA format is named: "<<name<<"\n";
        fout.open(name);

        // network files for input
        fin2.open(fname);
        done = false;
        if(!fin2) {
                std::cout << "\n\ncannot open " << fname << "\n";
                done = true;
        }
      buildNetwork(ip,a,j00,B,Binv,BSize,eps,done,ieee,FractOverload,
                   nodes,ArSize,Paveraged,lines,Nlines,rhs);
      countGenerators(j00,generatorNo,nPg,a,izrov,iposv,ArSize,B,
             BSize,Binv,M,N,nodes,Ptotal0,deltaP,Nlines,lines,rhs,ip);
      labelOutputFiles();
      return 0;
} // initializeSystem

int finalizeSystem()
{
    outputSummary();

    // Build ReStart file
    ReStart();

    stop_time = clock();

    std::cout << "\nTime taken = " << (stop_time - start_time) /
                 (CLOCKS_PER_SEC * 60.) << " min. \n";
    std::cout << "Program ends" << "\a";

    std::cout << "\nHello, World!\n";
    exitRoutine(0);
    return 0;
} // finalizeSystem()

void getInputParameters(int & ieee,int & start,int & AnaTime,
                        double & lambda, double & gamma_reg,
                        double & mu,double & p0,double & p1,
                        double & zmult,double & eps,
                        double & CritMargin,double & incrGenP,
                        int & days_gen,double & lmult,int & cc)
{
        /*****************************
                Input time evolution params
        *****************************/
        std::cout << "\n\t OPA code ";
        std::cout << "\n\t Random fluctuation of loads by regions ";
        std::cout <<
      "\n\t (letter in front of node name in file determines region) ";
        std::cout << "\n\t Overloaded lines are upgraded ";
        std::cout << "\n\t Input file name for power grid: ";
        std::cin  >> fname;
        std::cout << fname;
        std::cout <<"\n\tFormat of file(0=OPA,1=IEEE Common Format): ";
        std::cin  >> ieee;
        std::cout << ieee;
```

```
        if (ieee==1){
              std::cout <<
          "\n\t Input file name for line information in IEEE format: ";
              std::cin        >> fname2;
              std::cout       << fname2;
        }
        std::cout <<
     "\n\tNumber of time steps before system is in steady state with no output to files:
";
        std::cin  >> start;
        std::cout << start;
        std::cout <<
"\n\tNumber of time steps written to output file for statistical analysis: ";
        std::cin  >> AnaTime;
        std::cout << AnaTime;
        TotalTime = AnaTime + start;
        std::cout <<"\n\tDaily multiplier: demand for power (lambda=1.00005): ";
        std::cin  >> lambda;
        std::cout << lambda;
        std::cout << "\n\tDaily load fluctuation (gamma=1.67, maximum value of 2): ";
        std::cin  >> gamma_reg;
        std::cout << gamma_reg;
        std::cout << "\n\tMultiplier for upgrading of a failed line if it resulted in a
blackout (mu=1.007): ";
        std::cin  >> mu;
        std::cout << mu;
        std::cout << "\n\tProbability of an outage from a random event (p0=0 for none):
";
        std::cin  >> p0;
        std::cout << p0;
        std::cout << "\n\tProbability for an overloaded line to undergo an outage (p1=0
for none): ";
        std::cin  >> p1;
        std::cout << p1;
        /*************************************
                Uncomment the next 6 lines if you want to
                vary zmult from 1000 or eps from 0.1
        *************************************/
        /*************************************
        std::cout << "\n\tMultiplier for impedence of line (z) if line fails (zmult=1000)
=  ";
        std::cin  >> zmult;
        std::cout << zmult;
        std::cout << "\n\tEps (distance from maximum flow before a line is considered
overloaded) for overload flagging (eps=0.1) =  ";
        std::cin  >> eps;
        std::cout << eps;
        *************************************/
        std::cout << "\n\tNormalized minimal generator capability margin (delta_P/P=0.3):
";
        std::cin  >> CritMargin;
        std::cout << CritMargin;
        std::cout << "\n\tQuantized increase in generator power (kappa=0.04): ";
        std::cin  >> incrGenP;
        std::cout << incrGenP;
        std::cout << "\n\tDays to build new generators (days_gen>number of time steps for
no improvement): ";
        std::cin  >> days_gen;
        std::cout << days_gen;
        std::cout << "\n\tMultiply initial loads by this constant (lmult=1): ";
        std::cin  >> lmult;
        std::cout << lmult;
        std::cout << "\n\tTo reproduce this run input this seed (remove commented input
of cc in source): ";
/*************************************
        Enable std::cout for seed from clock,std::cin manual seed entry
        *************************************/
        std::cout << cc << "\n";
/*************************************
```

```
        Remove comment from next line to manually input seed
        ************************************/
        std::cin  >> cc;
}


void buildNetwork(int *&ip,double **&a,int *&j00,double **&B,
          double **&Binv,int& BSize,double& eps,bool& done,int& ieee,
          double *&FractOverload,Node*& nodes,int& ArSize,
          double *&Paveraged,PLine *&lines,int& Nlines,double *&rhs)
{
        /*****************************
                Build Network
        *****************************/
        int i = 0;
        while(!done) {
                nodes[i] = Build(fin2, done);
                i++;
        }
        if(ieee==0) {
                int l = 0;
                for ( int i = 0; i < ArSize ; i++) {
                        Paveraged[i]= 0.0;
                        for ( int k = 0; k < nodes[i].conN ; k++)    {
                                if (i<nodes[i].xin[k]) {
                                        lines[l] =Build(fin2,i,nodes[i].xin[k]);
                                        FractOverload[l]=0;
                                        l++;
                                }
                        }
                }
                Nlines = l;
        } else if(ieee==1) {
                for ( int i = 0; i < ArSize ; i++) {
                        Paveraged[i]= 0.0;
                        nodes[i].conN=0;
                        nodes[i].zone=3;
                        if(i<ArSize/3)nodes[i].zone=1;
                        else if(i<2*ArSize/3)nodes[i].zone=2;
                }
                int l=0;
                int one=0,two=0;
                fin2.close();
                fin2.open(fname2);
                if(!fin2) {
                        std::cout << "\n\ncannot open " << fname2 << "\n";
                        done = true;
                }
                while (fin2) {
                        lines[l] =Build(fin2,one,two);
                        FractOverload[l]=0;
                        for (int l0 = 0; l0 < l; l0++)
                                if((lines[l].Node1==lines[l0].Node1 &&
                                    lines[l].Node2==lines[l0].Node2)||
                                   (lines[l].Node1==lines[l0].Node2 &&
                                    lines[l].Node2==lines[l0].Node1))
                                { //duplicate lines
                                 lines[l0].Z=1./(1./lines[l0].Z+1./lines[l].Z);
                                 lines[l0].Zinit=lines[l0].Z;
                                 l--;
                                }
                        l++;
                }
            Nlines=l-1;
            calculateBhat(ArSize,B,Binv,BSize,ip,eps,Nlines,
                          lines,nodes,rhs);
    }
} // end buildNetwork()
```

```
int calculateBhat(int &ArSize,double **&B,double **&Binv,
                  int & BSize,int *&ip,double &eps,int &Nlines,
                  PLine *&lines,Node *&nodes,double *&rhs)
{
   //Calculate Bhat and Bhat-inverse
   int ier=Bhat(BSize,Nlines,lines,B,Binv,rhs,ip);
   if (ier != 0) {
   std::cout <<"Error in Matrix Decomposition row=\t"<<ier<<"\n";
   return ier ;
   }

   int overload;
   overload=flows(eps,ArSize,Nlines,lines,nodes,B,rhs,ip);
   for (int l = 0; l < Nlines; l++) {
        lines[l].LFmax=ceil(fabs(lines[l].Flow))+0.2;
        lines[l].Fmax=lines[l].LFmax;
        if(nodes[lines[l].Node1].Type == 2 ||
                                  nodes[lines[l].Node1].Type == 3)
         nodes[lines[l].Node1].PowerLimit+= floor(lines[l].LFmax);
        if(nodes[lines[l].Node2].Type == 2 ||
                                  nodes[lines[l].Node2].Type == 3)
         nodes[lines[l].Node2].PowerLimit+=
       floor(lines[l].LFmax);
        nodes[lines[l].Node1].conN++;
        nodes[lines[l].Node1].xin[nodes[lines[l].Node1].
                                        conN-1]=lines[l].Node2;
        nodes[lines[l].Node2].conN++;
        nodes[lines[l].Node2].xin[nodes[lines[l].Node2].
                                        conN-1]=lines[l].Node1;
       if(nodes[lines[l].Node1].conN>10||
                                  nodes[lines[l].Node2].conN > 10)
       {
                std::cout << "Increase i in function Build for dimensioning number of
neighbors for ieee==1 to more than max of " <<    nodes[lines[l].Node1].conN <<" and "<<
nodes[lines[l].Node2].conN << "\n";
                return 1;
        }
   }
   return 0; // should never reach this far
}

void countGenerators(int *&j00,int *&generatorNo,int& nPg,double **&a,int *&izrov,int
*&iposv,int &ArSize,double **&B,int& BSize,double **&Binv,int& M,int &N,Node
*&nodes,double &Ptotal0,double &deltaP,int &Nlines,PLine *&lines,double *&rhs,int *&ip)
{
        /*****************************
                Count number of generators
        *****************************/
        nPg = 0;
        for (int i = 0; i < ArSize; i++)
                if (nodes[i].Type == 2 ||nodes[i].Type == 3) nPg++;
        generatorNo = new int[nPg];
        int j = 0;
        for (int i = 0; i < ArSize; i++)
                if (nodes[i].Type == 2 || nodes[i].Type == 3)
                {
                   generatorNo[j]=i;
                   j++;
              }
        int k = nPg;
        for (int i = 0; i < nPg; i++)
        {       int j=(int) (rand()/g) * k;
                //Scramble generator numbers
             if( j < k)Swap(generatorNo+j,generatorNo+(k-1));
                k--;
        }
        Ptotal0 = 0;

        //Initial Power of loads in the network
```

```
        for (int i = 0; i < ArSize; i++)Ptotal0 -= nodes[i].Power;
        deltaP = 0;
        for (int i = 0; i < ArSize; i++)deltaP += nodes[i].PowerLimit;
        deltaP=deltaP/Ptotal0-1;             // closeness to maximum power
        std::cout << "delta_P/P for input file = \t"<< deltaP << "\n";

        /*******************************
               New arrays for simplex method
        *******************************/
        M = 2*Nlines+1; //+nPg;
        N = 0;
        j00 = new int[2*ArSize];
         for (int i = 0; i < ArSize; i++) {
               if (nodes[i].Type ==1 || nodes[i].Type ==2) {
                      j00[N]=i;
                      N++;
               } else if(nodes[i].Type ==3) {
                      j00[N]=i;
                      j00[N+1]=i;
                      N = N+2;
               }
        }

        M=M+N;
        a=matrix(1,M+2,1,N+1);
        izrov = ivector(1,N);
        iposv = ivector(1,M);

        /*********************************
               Calculate Bhat and Bhat inverse
        *********************************/
        int ier=Bhat(BSize,Nlines,lines,B,Binv,rhs,ip);
        if (ier != 0) {
           std::cout << "Error in Matrix Decomposition row: \t"<<
             ier << "\n";
          exitRoutine(ier);
        }
}

void labelOutputFiles()
{
        /****************************
               Labels for output files
        ****************************/
        fout5 << "lines_or_nodes" << "\t" << "FractOverload" << "\t" << "PDFoutages" <<
"\t" << "Paveraged" << "\t" << "PDFblackouts" << "\n";
        fout7 << "time" << "\t" << "Load Shed" << "\t" << "Demand" << "\t" << "Power
Served" << "\t" << "DP/P" << "\t" << "Total Demand-Power Served" << "\t" << "Load
shed/Power Demand" << "\t" << "<M>" << "\t" << "<Delta_F>" << "\t" << "Total overloads"
<< "\t" << "outages"
                                << "\t" << "iterations\tSum Fmax" << "\n";
        fout2 << "time\tp0 outages";
        for (int i = 0; i < 10; i++)fout2 << "\titeration" << i+1;
        fout2 << "\n";
        fout4 << "time";
        for (int i = 0; i < 10; i++)fout4 << "\titeration" << i+1;
        fout4 << "\n";
}

int timeLoop()
{
        RealTime++;
        //if(time_d%10000==0) std::cout << "Time = "<< time_d << "\n";
        if(time_d%100==0) std::cout << "Time = "<< time_d << "\n";

        //Restore Zij and Fmax
        for (int l = 0; l < Nlines; l++) {
               lines[l].Fmax=lines[l].LFmax;
               lines[l].Z=lines[l].Zinit;
```

```
 }

 //Restore power shed
 for (int i = 0; i < ArSize; i++) {
        nodes[i].PowerShed=0;
        nodes[i].Demand = nodes[i].Power;
 }

 //Random change of loads
 if (RealTime > 1)
{
        Ptotal = 0;  //Total Power of loads in the network
        for (int i = 0; i < 123; i++) xran[i]=rand()/g;
        for (int i = 0; i < ArSize; i++)
     {
              nodes[i].PowerAv*=lambda;
       // increase power of all nodes
             nodes[i].Power   = nodes[i].PowerAv;
          // by regular growth lambda
              Ptotal -= nodes[i].Power;
              // 2-gamma to gamma
              nodes[i].Power *= (2*(gamma_reg-1)
               *xran[nodes[i].zone]+2-gamma_reg);
              nodes[i].Demand = nodes[i].Power;
        }

        //Improve Fmax due to overload and blackouts, and outages
        countY       = 0;
        for (int l = 0; l < Nlines; l++)
       {
              // if blackout improve overlaoded lines
              if(lines[l].Overload > 0 && blackout > 0) {
                     lines[l].Fmax*=mu; // by factor mu
                     lines[l].LFmax = lines[l].Fmax;
                     lines[l].Out = false;
              }

              // Calculate random new outages
              if ( rand()/g > 1-p0) {
                     lines[l].Fmax = lines[l].LFmax/zmult;
       // divide Fmax by zmult
                     lines[l].Z*=zmult;// multiply Fmax by zmult
                     countY++;      // count # of lines down this
                            // time_d step
                     lines[l].Out1 = 1;// first outage of blackout
              }
       }

       //Change of Zij needs new Bhat and Bhat-inverse
       if (countY != 0 || iterations != 0) {
              int ier=Bhat(BSize,Nlines,lines,B,Binv,rhs,ip);
              if (ier != 0) {
                 std::cout <<"Error in Matrix Decomposition
                                          row=\t"<<ier<<"\n";
                 return ier ;
              }
       }
 } // if (RealTime > 1)

 //While there are overloads, cause outages by p1
 iterations = 0;
 outages = countY;       // include outages caused by p0
 overloadTotal = 0; // total # of overloads in this time_d step
 PowerShedTotal = 0;// total power shed in one iteration
 icount = 0;    // # of outages due to overloads in one iteration
 icountm1 = 0; // # of outages due to overloads in last iteration
 for (int l = 0; l < Nlines; l++)lines[l].Overload=0;
 TotalSumFmax = 0;
int ier;
```

```
       do
      {
          ier = innerTimeStepIterations();
       }while(icount > 0 && ier == 0); // while not steady state

      if(time_d > start && blackout==1 ) {
               fout2 << RealTime;
               for (int i = 0; i <= iterations;i++)
                       fout2<<"\t"<<Out_iter[i];
               for (int i = iterations+1; i <= 10; i++)fout2 <<"\t"<<0;
               fout2 << "\n";
               fout4 << RealTime;
               for (int i = 1; i <= iterations; i++)
                       fout4 <<"\t"<<PS_iter[i];
               for (int i = iterations+1; i <= 10; i++)fout4 <<"\t"<<0;
               fout4 <<"\n";
       }
       if(time_d%(days_gen)==0) {
               for (int i = 0; i < ArSize; i++)if (nodes[i].Type == 2 || nodes[i].Type ==
3) {
                               nodes[i].Q=0;
                               for (int l = 0; l < Nlines; l++)if(lines[l].Node1 ==i ||
lines[l].Node2 ==i)nodes[i].Q +=lines[l].Fmax;
                               nodes[i].deltaQ=incrGenP*nodes[i].PowerLimit;
                       }
                       int k=nPg;
                       for (int i = 0; i < nPg; i++) {
                               int j=(int) (rand()/g) * k;
                               if( j < k)Swap(generatorNo+j,generatorNo+(k-1)); //Scramble
generator numbers
                               k--;
                       }
                       for (int j = 0; j < nPg; j++) {
                               double SumLimit=0;
                               for (int i = 0; i < ArSize; i++)SumLimit +=
nodes[i].PowerLimit;
                               deltaP=SumLimit/(Ptotal0*exp((lambda-1)*time_d))-1;
       // closeness to maximum power
                               if(deltaP - CritMargin <= 0 &&
nodes[generatorNo[j]].PowerLimit > 0) {
                                       if(nodes[generatorNo[j]].deltaQ <=
nodes[generatorNo[j]].Q - nodes[generatorNo[j]].PowerLimit) {
                                               nodes[generatorNo[j]].PowerLimit +=
nodes[generatorNo[j]].deltaQ ;
                                       }
                               }
                       }
               }

   /*********************************
   Calculate Diagnostics
   *********************************/
   for ( int l = 0; l < Nlines ; l++) {
                       FractOverload[l]=(RealTime*FractOverload[l] +
lines[l].MFlow1)/(RealTime+1);
   }

   for (int i = 0; i < ArSize; i++) {
                       Paveraged[i]=(RealTime*Paveraged[i] +
(nodes[i].Generator+nodes[i].Power)/pow(lambda,RealTime))/(RealTime+1);
                       if((nodes[i].Power/nodes[i].Demand < 0.1) && (nodes[i].Demand !=
0))
                       nodes[i].PDFblackouts++;
   }

   // Time_var diagnostics
   if ( blackout > 0 && time_d>start) {
                       double Mavg=0;
                       double Foverload=0;
```

```
                        double F=0;

                        // add first outages to PDFoutages
                        for (int l = 0; l < Nlines; l++) {
                                lines[l].PDFoutages += lines[l].Out1;
                                lines[l].Out1 = 0;
                        }

                        for ( int l = 0; l < Nlines ; l++) {
                                Mavg+=fabs(lines[l].Flow)/lines[l].LFmax;
                                if(lines[l].Overload > 0 && blackout > 0)
                                Foverload+=lines[l].LFmax;
                                F+=lines[l].LFmax;
                        }
                        deltaP = 0;
                        for (int i = 0; i < ArSize; i++)deltaP += nodes[i].PowerLimit;
                        deltaP = deltaP/(Ptotal0*exp((lambda-1)*time_d))-1;         //
closeness to maximum power

                        fout7 << RealTime << "\t" << PowerShedTotal << "\t" << Dtotal <<
"\t" << Ptotal << "\t" << deltaP << "\t" << Dtotal-Ptotal
                                                << "\t" << PowerShedTotal/Dtotal << "\t" <<
Mavg/Nlines << "\t" << Foverload/F << "\t" << overloadTotal << "\t"
                                                << outages << "\t" << iterations << "\t" <<
TotalSumFmax << "\n";
   } // if (blackout...)

   //Reset data if start time_d
   if (time_d == start) {
        RealTime = 0;
        // reset line counters
        for ( int l = 0; l < Nlines ; l++) {
                FractOverload[l] = 0;
                lines[l].PDFoutages = 0;
        }
        // reset load counters
        for ( int i = 0; i < ArSize; i++)     {
                Paveraged[i]= 0.0;
                nodes[i].PDFblackouts = 0;
        }
   }
   time_d++;
   return time_d;
} //end timeLoop

void outputSummary()
{
        // Write to "Averaged"
        AnaTime=time_d-1-start;
        for (int i = 0; i < ArSize; i++)
        Tblackouts += nodes[i].PDFblackouts;
        for (int l = 0; l < Nlines; l++) {
                Toutages += lines[l].PDFoutages;
        }
        int min=ArSize;
        if(Nlines < min) min=Nlines;
        for (int i = 0; i < min; i++) {
                AVblackouts = nodes[i].PDFblackouts/Tblackouts;
                AVoutages = lines[i].PDFoutages/Toutages;
                fout5 << i << "\t" << FractOverload[i] << "\t" << AVoutages << "\t" <<
Paveraged[i] << "\t" << AVblackouts << "\n";
        }

        for (int l = ArSize; l < Nlines; l++) {
                AVoutages = lines[l].PDFoutages/Toutages;
                 //i = (double) l;
                fout5 << l << "\t" << FractOverload[l] << "\t" << AVoutages << "\n";
        }
```

```
        for (int l = Nlines; l < ArSize; l++) {
                AVoutages = lines[l].PDFoutages/Toutages;
                fout5 << l << "\t" << "\t" << "\t" << Paveraged[l] << "\t" << AVblackouts
<< "\n";
        }
}

int innerTimeStepIterations()
{

    PowerShedIter = 0;
    //Calculate A matrix
    Amatrix(M,N,ArSize,Nlines,lines,nodes,a,Binv);

    /****  SIMPLEX solver  ****/
    //int m1=M-1-nPg;

    //2*Nlines+#Nodes <= inequalities  nPg >= inequalities     1 equality
    simplx(a,M,N,M-1,0,1,&icase,izrov,iposv);//simplx(a,M,N,M-1-
nPg,nPg,1,&icase,izrov,iposv);

    //std::cout <<"icase="<< icase<<"\t";
    if (icase != 0) {
           std::cout<<"Total blackout"<< time_d << "\t"<< iterations << "\n";
           for (int i = 0; i < ArSize; i++) {
               nodes[i].PowerShed=-nodes[i].Power;
               nodes[i].Power=0;
               nodes[i].Generator=0;
           }
    }
    else {
        //Put results of SIMPLEX solver into Power array
        // calculate power and power shed for all loads
        for (int j = 1; j <= M; j++)if(iposv[j]<=N) {
             int j1=j00[iposv[j]-1];
             int j2=-1;
             if (iposv[j] >= 2)j2=j00[iposv[j]-2];
             if(nodes[j1].Type == 2 || (nodes[j1].Type == 3 && j1 != j2))
             {
                    nodes[j1].Generator=a[j+1][1];
             }
             if(nodes[j1].Type == 1 || (nodes[j1].Type == 3 && j1 == j2))
             {
                    nodes[j1].PowerShed=-nodes[j1].Power-a[j+1][1];
                    nodes[j1].Power=-a[j+1][1];
             }
        }
        for (int j = 1; j <= N; j++)if(izrov[j]<=N)
        {
             int j1=j00[izrov[j]-1];
             int j2=-1;
             if (izrov[j] >= 2)j2=j00[izrov[j]-2];
             if(nodes[j1].Type == 2 || (nodes[j1].Type == 3 && j1 != j2)) {
                    nodes[j1].Generator=0;
             }
             if(nodes[j1].Type == 1 || (nodes[j1].Type == 3 && j1 == j2)) {
                    nodes[j1].PowerShed=-nodes[j1].Power;
                    nodes[j1].Power=0;
             }
        }
    } // else
    Ptotal = 0;  //Total Power of loads in the network
    for (int i = 0; i < ArSize; i++)
         Ptotal -= nodes[i].Power;               // calculate total power of all loads
    Dtotal  = 0;  //Total Power of loads in the network
    for (int i = 0; i < ArSize; i++)
         Dtotal -= nodes[i].Demand;              // calculate total power of all loads
    for (int i = 0; i < ArSize; i++) {
```

```
         PowerShedTotal+=nodes[i].PowerShed;     // calculated total power shed of all
loads
         PowerShedIter+=nodes[i].PowerShed;
    }

    if(PowerShedTotal/Dtotal>1e-5)blackout=1;  // determine if blackout
    else blackout=0;

    //diagnostics for step
    //Calculate flows
    int overload=flows(eps,ArSize,Nlines,lines,nodes,B,rhs,ip);
    overloadTotal+=overload;
    if(iterations==0)for (int l = 0; l < Nlines;
l++)lines[l].MFlow1=fabs(lines[l].Flow)/lines[l].LFmax;
    iterations++;

    //Random outages due to overloads
    icount = 0;
    SumFmax = 0;
    if(iterations==1) {
        icountm1=outages;
        SumFmaxm1=0;
        for (int l = 0; l < Nlines; l++)if(lines[l].Fmax !=
lines[l].LFmax)SumFmaxm1+=lines[l].LFmax;
               SumFmaxm1/=Ptotal0*exp((lambda-1)*time_d);
    }
    for (int l = 0; l < Nlines; l++)if(fabs(lines[l].Flow) >lines[l].LFmax-eps) {
        if ( rand()/g > 1-p1) {
               lines[l].Fmax = lines[l].LFmax/zmult;   // decrease Fmax
               lines[l].Z*=zmult;
// increase Z
               outages++;
// count # of outages this time_d step
               icount++;
// count # of outages this iteration
               lines[l].Out = true;
// set outage flag
               SumFmax+=lines[l].LFmax;
               if (iterations == 1)
                    lines[l].Out1 = 1;
// first outage of blackout
         }
    }

    //Change of Zij needs new Bhat and Bhat-inverse
    int ier = 1;
    if (icount != 0) {
       ier=Bhat(BSize,Nlines,lines,B,Binv,rhs,ip);
       if (ier != 0)
       {
            std::cout <<"Error in Matrix Decomposition row=\t"<<ier<<"\n";
            exitRoutine(ier) ;
       }
    }
    SumFmax/=Ptotal0*exp((lambda-1)*time_d);
    if(time_d > start) {
       if(iterations==1) {
           Out_iter[0]=outages;
           Out_iter[1]=icount;
           PS_iter[1]=PowerShedIter/Dtotal;
       }
       if(iterations>1 && iterations<=10) {
           Out_iter[iterations]=icount;
           PS_iter[iterations]=PowerShedIter/Dtotal;
       }
    }
    icountm1=icount;
    SumFmaxm1=SumFmax;
    TotalSumFmax+=SumFmax;
```

```
    return ier;
}
void exitRoutine(int input)
{
        // Close all files
        fin.close();
        fin2.close();
        fout.close();
        fout5.close();
        fout7.close();
         std::cout << "OPA Program finished with " << input << " as its exit code\n";
         //exit(input);
         return;
}

double getRAND_MAX()
{ return (double) RAND_MAX; }

int getNextRand()
{ return rand(); }


/*************************************************************
 *  Build

 *

 *      Build the network:  this version corresponds to a ring
 *      network with only one connection in and one out of each
 *  node.

 *

 *  called by:

 *              main

 *              ReStart

 *************************************************************/
Node    Build(ifstream& f, bool& done) {
        int Xpos, Ypos;
        double iz;
        double izL=0;
        double dummy;
        int type;
        double gen;
        char c0, c1, c2;
        char name[20];
        int i = 0;
        bool line = false;
        int vin[20];                            // connections in

        Node buildnode;
        if(ieee==1) {
                i=10; //increase if codes stops with dimensioning problem
                Xpos=100;
                Ypos=100;
                f
>>dummy>>name>>c0>>dummy>>dummy>>dummy>>dummy>>dummy>>dummy>>iz>>dummy>>gen>>dummy>>dummy
>>dummy>>dummy>>dummy>>dummy>>dummy>>dummy;
                iz*=-0.01;
                gen*=0.01;
                type=0;
                if(iz!=0)type=1;
                if(gen!=0)type+=2;
                if(f.eof())done = true;
        } else {
                // get iz#
```

```
            f.get(c0);
            while(isspace(c0))
            f.get(c0);

            if(c0 == '/'){                          // check for end of file
                    done = true;
                    buildnode = Node();
                    f.get(c1);
                    return buildnode;
            }
            f >> iz;
  f.get(c1);
  if(c0 == '$') {
                    type = (int) iz;
                    f >> iz;
                    if (type ==1 || type ==3)iz*=lmult;
    f.get(c1);
    f >> gen;
    f.get(c1);
    f >> izL;
    izL*=lmult;
    f.get(c1);
            } else {
                    izL     = 0;
                    type=1;
                    gen=0;
                    if(iz > 0) {
                            izL= 1.3*iz;
                            gen=iz;
                            iz=0;
                            type=2;
                    }
            }

     // get name
  f.get(c1);
  while(c1 != ' ') {
                    name[i] = c1;
    f.get(c1);
    i++;
            }
  name[i] = '\0';
            f.get(c1);                                      // get " ("

  // get Xpos
  f >> Xpos;
            f.get(c1);                                      // get ", "
  f.get(c1);

  // get Ypos
  f >> Ypos;
  f.get(c1);                                // get ") ["
  f.get(c1);
  f.get(c1);

  // get neighbors
     i = 0;
            while(!line) {
                    f >> vin[i];
    f.get(c1);
    f.get(c2);
    if(c1 == ']')
                            line = true;
      i++;
                    }
     } // if ieee==1
buildnode = Node(Xpos, Ypos, i, iz, izL, gen, type, vin, name);

return buildnode;
```

```
} // Node Build()

/*************************************************************
 *  ReStart

 *

 *      Build the ReStart network file.

 *

 *  called by: main

 *

 *  calls: Build

 *************************************************************/
void ReStart() {
        // write file
        char    c;
        for (int i = 0; i < ArSize; i++) {
                c='a';
                if(nodes[i].zone==2)c='b';
                else if(nodes[i].zone==3)c='c';
                fout << "$"<< nodes[i].Type<< " " << nodes[i].PowerAv << " " <<
nodes[i].Generator << " " << nodes[i].PowerLimit
                                << " " << c << i << " ("<< nodes[i].locX << ", " <<
nodes[i].locY << ") [";
                for (int k = 0; k < nodes[i].conN-1; k++) fout << nodes[i].xin[k] << ", ";
                fout << nodes[i].xin[nodes[i].conN-1] << "]\n";
        }
        fout << "\n//\n";
        for (int l = 0; l < Nlines; l++)
        fout << lines[l].Node1 << "\t" << lines[l].Node2 << "\t" << lines[l].Zinit << "\t"
<< lines[l].LFmax << "\t" << 1 << "\n";
}


/*************************************************************/

PLine Build(ifstream& f,int one, int two) {
        PLine  buildline;
        double iz=1.;
        double fmax=3.;
        double rf = 1.;
        if(ieee==1) {
                f >> one >> two >> rf >> rf >> rf >> rf >> rf >> iz >> rf >> rf >> rf >>
rf >> rf >> rf >> rf >> rf >> rf >> rf >> rf >> rf >> rf >> rf ;
                one--;
                two--;
        }
        else if (!f.eof()) f >> one >> two >> iz >> fmax >> rf;
        buildline = PLine(one,two,iz,fmax);
        return buildline;
}

/*************************************************************
 *      NodeCount

 *

 *      Count the number of nodes in a network.
 *

 *      called by: main

 *************************************************************/
```

```
int NodeCount(ifstream& f) {
	int Count = 0;
	char c = '\0';
	if(ieee==1) {
		while(f){
			f >> Count;
			f.ignore(2000,'\r');   // end of line
			f.ignore(2000,'\r');   // end of line
		}
	} else {
		while(c != '/'){
			if((c == '#') || (c == '$'))
			Count++;
			f.get(c);
		}
	}
	return Count;
}

/*************************************************************
 *  flows

 *

 *  Caluclates the Power Flows in each line
 *

 *  called by: main

 *************************************************************/

int flows(double eps,int ArSize,int Nlines,PLine *lines,Node *nodes,double **B,double
*rhs, int *ip) {
	int BSize=ArSize-1;
	for (int i = 0; i < BSize; i++)rhs[i]=nodes[i+1].Generator+nodes[i+1].Power;

	sol(BSize,B,rhs,ip);
	//theta=B-inverse*Power
	for ( int i = 1; i < ArSize ; i++)nodes[i].theta=rhs[i-1];
	nodes[0].theta=0;

	int overload=0;
	for ( int l = 0; l < Nlines ; l++) {
		lines[l].Flow=(nodes[lines[l].Node1].theta-
nodes[lines[l].Node2].theta)/lines[l].Z;
		//overloads defined with eps
		if(fabs(lines[l].Flow) >lines[l].LFmax-eps) {
			overload++;
			lines[l].Overload=1;
		}
	}
	return overload;
}

/*************************************************************
 *  Bhat

 *

 *  Calculates Bhat matrix

 *

 *  called by: main

 *

 *  calls: sol
```

```
 ************************************************************/

int Bhat(int BSize,int Nlines, PLine *lines,double **B, double **Binv,double *rhs, int
*ip) {
       for (int i = 0; i < BSize; i++) {
              for (int j = 0; j < BSize; j++) {
                     B[i][j]=0.;
              }
       }
       //Calculate B matrix
       for ( int l = 0; l < Nlines ; l++) {
              if(lines[l].Node1>0)
              B[lines[l].Node1-1][lines[l].Node1-1]+=1/lines[l].Z;
              if(lines[l].Node2 > 0)
              B[lines[l].Node2-1][lines[l].Node2-1]+=1/lines[l].Z;
              if(lines[l].Node1>0 && lines[l].Node2 > 0) {
                     B[lines[l].Node2-1][lines[l].Node1-1]=-1/lines[l].Z;
                     B[lines[l].Node1-1][lines[l].Node2-1]=-1/lines[l].Z;
              }
       }
       /*ofstream foutb("Bmatrix");
       for (int i = 0; i < BSize; i++)
       {
       for (int j = 0; j < BSize-1; j++)foutb<<B[i][j]<<"\t";
       foutb<<B[i][BSize-1]<<"\n";
       }*/

       //Calculate B-inverse from B*B-inverse=I
       int ier=dec(BSize,B,ip);
       if (ier != 0)return ier;
       for (int j = 0; j < BSize; j++) {
              for (int i = 0; i < BSize; i++)rhs[i]=0.;
              rhs[j]=1.;
              sol(BSize,B,rhs,ip);
              for (int i = 0; i < BSize; i++)Binv[i][j]=rhs[i];
       }
       return 0;
}

/************************************************************
 *  Amatrix

 *

 *  Calculates A matrix

 *

 *  called by: main

 ************************************************************/

void Amatrix(int M, int N,int ArSize,int Nlines,PLine *lines,
Node *nodes,double **a,double **Binv) {
       for (int i = 1; i <= M+2; i++) {
              for (int j = 1; j <= N+1; j++) {
                     a[i][j]=0;
              }
       }
       int j=0;
       for (int j00 = 0; j00 < ArSize; j00++) {
              if (nodes[j00].Type ==2 || nodes[j00].Type ==3) {
                     //Function to maximize (Generators)
                  a[1][j+2]=-1;
                     //Generators <= Power limit
                     a[2+2*Nlines+j][j+2]=-1;
                     //Sum of powers = 0 (Generators)
                a[2+2*Nlines+N][j+2]=1;
                     //Flows of lines <=Fmax
```

```
                for (int i = 0; i < Nlines; i++) {
                                if (j00!=0) {
                                        if( lines[i].Node1 !=
0)a[i+2][j+2]=1./lines[i].Z*(Binv[lines[i].Node1-1][j00-1]);
                                        if( lines[i].Node2 != 0)a[i+2][j+2]-
=1./lines[i].Z*(Binv[lines[i].Node2-1][j00-1]);
                                        a[i+2+Nlines][j+2]=-a[i+2][j+2];
                                }
                        }
                  j++;
                }
                if (nodes[j00].Type ==1 || nodes[j00].Type ==3) {
                        //Function to maximize (Sinks)
                  a[1][j+2]=100;
                        //Sinks <= Initial power of Sinks
                        a[2+2*Nlines+j][j+2]=-1;
                        //Sum of Powers = 0 (Sinks)
                  a[2+2*Nlines+N][j+2]=-1;
                        //Flows of lines <=Fmax
                  for (int i = 0; i < Nlines; i++) {
                                if (j00!=0) {
                                        if( lines[i].Node1 !=
0)a[i+2][j+2]=1./lines[i].Z*(Binv[lines[i].Node1-1][j00-1]);
                                        if( lines[i].Node2 != 0)a[i+2][j+2]-
=1./lines[i].Z*(Binv[lines[i].Node2-1][j00-1]);
                                        a[i+2][j+2]=-a[i+2][j+2];
                                        a[i+2+Nlines][j+2]=-a[i+2][j+2];
                                }
                        }
                  j++;
                }
        }
        //Function to maximize (RHS)
        a[1][1]=0;
        //Flows of lines <=Fmax (RHS)
        for (int i = 0; i < Nlines; i++) {
                a[i+2][1]=lines[i].Fmax;
          a[i+2+Nlines][1]=lines[i].Fmax;
        }
 // int k=0;
        //Sinks <= Initial power of Sinks (RHS)
        int i=0;
        for (int j00 = 0; j00 < ArSize; j00++) {
                if (nodes[j00].Type ==2 || nodes[j00].Type ==3) {
                        a[2+2*Nlines+i][1]=nodes[j00].PowerLimit;
                        i++;
                }
                if (nodes[j00].Type ==1 || nodes[j00].Type ==3) {
                        a[2+2*Nlines+i][1]=-nodes[j00].Power;
                        i++;
                }
        }

        //Sum of Powers = 0 (RHS)
        a[2+2*Nlines+N][1]=0;

        /*ofstream fout("Anew");
        fout<<"A:\n";
        for (int i = 1; i <= M+2; i++)
    {
        for (int j = 1; j <= N+1; j++)
    fout <<a[i][j]<<"\t";
    fout<<"\n";
        }*/
        return;
}

/*****************************************************************/
/*************************************************************
```

```
 *  simplx

 *

 *  This is the big simplex slover
 *

 *  called by: main

 *

 *  calls:

 *      ivector, free_ivector
 *              simp1, simp2, simp3

 ************************************************************/
void simplx(double **a,int m,int n,int m1,int m2,int m3,int *icase,int izrov[],int
iposv[]) {
        int i,ip,ir,is,k,kh,kp,m12,nl1,nl2;
  int *l1,*l2,*l3;
  double q1,bmax;
  if (m!=(m1+m2+m3)) std::cout<<"bad input constraint counts in simplx\n";
  l1=ivector(1,n+1);
  l2=ivector(1,m);
  l3=ivector(1,m);
  nl1=n;
        for (k=1;k<=n;k++) {
                l1[k]=k;
    izrov[k]=k;
        }
  nl2=m;
  for (i=1;i<=m;i++) {
                if (a[i+1][1]<-EPS) {
                        *icase=-2;
            std::cout<< i <<" bad input tableau in simplx\n";
            return;
                }
    l2[i]=i;
    iposv[i]=n+i;
        }
  for (i=1;i<=m2;i++)l3[i]=1;
  ir=0;
  if (m2+m3) {
                ir=1;
    for (k=1;k<=(n+1);k++) {
                        q1=0.;
      for (i=m1+1;i<=m;i++) q1 +=a[i+1][k];
      a[m+2][k]=-q1;
                }
                do {
                        simp1(a,m+1,l1,nl1,0,&kp,&bmax);
      if (bmax<=EPS && a[m+2][1]<-EPS) {
                                *icase=-1;
        free_ivector(l3,1);
        free_ivector(l2,1);
        free_ivector(l1,1);
        //free_ivector(l3,1,m);
        //free_ivector(l2,1,m);
        //free_ivector(l1,1,n+1);
        return;
      } else if (bmax<=EPS && a[m+2][1]<=EPS){
                                m12=m1+m2+1;
        if (m12<=m){
                                        for(ip=m12;ip<=m;ip++) {
                                                if (iposv[ip]==(ip+n)){
                                                        simp1(a,ip,l1,nl1,1,&kp,&bmax);
              if (bmax>0.)goto one;
                                        }
```

```
                                        }
                                }
                                ir=0;
      --m12;
      if (m1+1<=m12)
      for (i=m1+1;i<=m12;i++)
      if (l3[i-m1]==1)
      for (k=1;k<=n+1;k++)
      a[i+1][k]=-a[i+1][k];
                                break;
                        }
    simp2(a,n,l2,nl2,&ip,kp,&q1);
    if (ip==0){
                                *icase=-1;
                                free_ivector(l3,1);
                                free_ivector(l2,1);
                                free_ivector(l1,1);
                                //free_ivector(l3,1,m);
                                //free_ivector(l2,1,m);
                                //free_ivector(l1,1,n+1);
                                return;
                        }
                        one: simp3(a,m+1,n,ip,kp);
    if (iposv[ip]>=(n+m1+m2+1)) {
      for (k=1;k<=nl1;k++)
      if (l1[k]==kp)break;
                                --nl1;
      for (is=k;is<=nl1;is++)l1[is]=l1[is+1];
      a[m+2][kp+1] += 1.0;
                                for (i=1;i<=m+2;i++) a[i][kp+1] = -a[i][kp+1];
    } else {
                                if (iposv[ip] >= (n+m1+1)) {
                                        kh=iposv[ip]-m1-n;
                                        if (l3[kh]) {
                                                l3[kh]=0;
                                                a[m+2][kp+1] += 1.0;
                                                for (i=1;i<=m+2;i++)
                                                a[i][kp+1]=-a[i][kp+1];
                                        }
                                }
                        }
    is=izrov[kp];
    izrov[kp]=iposv[ip];
    iposv[ip]=is;
             } while(ir);
     } // if (m2+m3)
for (;;) {
             simp1(a,0,l1,nl1,0,&kp,&bmax);
  if (bmax<=0.){
                        *icase=0;
    free_ivector(l3,1);
    free_ivector(l2,1);
    free_ivector(l1,1);
    //free_ivector(l3,1,m);
    //free_ivector(l2,1,m);
    //free_ivector(l1,1,n+1);
    return;
             }
  simp2(a,n,l2,nl2,&ip,kp,&q1);
             if (ip==0) {
                        *icase=1;
    free_ivector(l3,1);
    free_ivector(l2,1);
    free_ivector(l1,1);
    //free_ivector(l3,1,m);
    //free_ivector(l2,1,m);
    //free_ivector(l1,1,n+1);
    return;
             }
```

```
    simp3(a,m,n,ip,kp);
    is=izrov[kp];
    izrov[kp]=iposv[ip];
    iposv[ip]=is;
       }
}

/*************************************************************************/
void simp1(double **a,int mm,int ll[],int nll,int iabf,int *kp,double *bmax) {
       int k;
  double test;
  *kp=ll[1];
  *bmax=a[mm+1][*kp+1];
  for (k=2;k<=nll;k++){
              if (iabf == 0) test=a[mm+1][ll[k]+1]-(*bmax);
    else test=fabs(a[mm+1][ll[k]+1])-fabs(*bmax);
    if (test > 0.0) {
                        *bmax=a[mm+1][ll[k]+1];
      *kp=ll[k];
              }
       }
}

/*************************************************************************/
void simp2(double **a,int n,int l2[],int nl2,int *ip,int kp,double *q1) {
  int k,ii,i;
  double qp;
  double q0;
  double q;
  *ip=0;
  for (i=1;i<=nl2;i++)
  if (a[l2[i]+1][kp+1] < -EPS) break;
       if (i>nl2) return;
  *q1 = -a[l2[i]+1][1]/a[l2[i]+1][kp+1];
  *ip=l2[i];
  for (i=i+1;i<=nl2;i++) {
              ii=l2[i];
    if (a[ii+1][kp+1] < -EPS) {
                        q=-a[ii+1][1]/a[ii+1][kp+1];
      if (q < *q1) {
                              *ip=ii;
                              *q1=q;
      } else if (q == *q1) {
                              for (k=1;k<=n;k++) {
                                     qp=-a[*ip+1][k+1]/a[*ip+1][kp+1];
                                     q0=-a[ii+1][k+1]/a[ii+1][kp+1];
                                     if (q0 != qp) break;
                              }
                              if (q0 < qp) *ip=ii;
                        }
              }
       }
}

/*************************************************************************/
void simp3(double **a,int i1,int k1,int ip,int kp) {
       int kk,ii;
  double piv;
  piv=1.0/a[ip+1][kp+1];
       for (ii=1;ii<=i1+1;ii++)
       if (ii-1 != ip) {
              a[ii][kp+1] *= piv;
    for (kk=1;kk<=k1+1;kk++)
    if (kk-1 != kp)
    a[ii][kk]-=a[ip+1][kk]*a[ii][kp+1];
       }
  for (kk=1;kk<=k1+1;kk++)
  if (kk-1 != kp) a[ip+1][kk] *= -piv;
  a[ip+1][kp+1]=piv;
```

```
}

int *ivector(int nl,int nh) {
        int *v;
        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) std::cout<<"allocation failure in ivector()\n";
        return v-nl;
}

/*************************************************************************/
//void free_ivector(int *v,int nl,int nh)
void free_ivector(int *v,int nl) {
        free((char*) (v+nl));
}

/*************************************************************************/
double **matrix(int nrl,int nrh,int ncl,int nch) {
        int i;
  double **m;
  m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
  if (!m) std::cout<<"allocation failure 1 in matrix()\n";
  m-=nrl;
  for(i=nrl;i<=nrh;i++) {
              m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
    if (!m) std::cout<<"allocation failure 2 in matrix()\n";
    m[i]-=ncl;
        }
  return m;
}

/*************************************************************************/
//-------------------------------------------------------------------------
//  matrix triangularization by gauss elimination with partial pivoting.
//  input..
//     n = order of matrix.
//     ndim = declared first dimension of array  a.
//     a = matrix to be triangularized.
//  output..
//     a(i,j), i.le.j = upper triangular factor, u .
//     a(i,j), i.gt.j = multipliers = lower triangular factor, i - l.
//     ip(k), k.lt.n = index of k-th pivot row.
//     ier = 0 if matrix a is nonsingular, or k if found to be
//           singular at stage k.
//  row interchanges are finished in u, only partly in l.
//  use  sol  to obtain solution of linear system.
//  if ier .ne. 0, a is singular, sol will divide by zero.
//-------------------------------------------------------------------------
//
//  reference!  a. //. hindmarsh, l. j. sloan, k. w. fong, and
//              g. h. rodrigue,
//              dec/sol!  solution of dense systems of linear
//              algebraic equations,
//              lawrence livermore laboratory report ucid-30137,
//              june 1976.
//-------------------------------------------------------------------------
int dec (int n, double **a, int *ip) {
        int ier = 0;
  int k;
  if (n == 1) {
              k = n;
         if (a[n-1][n-1] == 0.) ier = k;
    return ier;
        }
  int nm1 = n - 1;
  for ( int k = 1; k <= nm1; k++) {
              int kp1 = k + 1;
              //  find the pivot in column k.  search rows k to n. --------------------
         int m = k;
         for ( int i = kp1; i <= n; i++)if (fabs(a[i-1][k-1]) > fabs(a[m-1][k-1])) m = i;
```

```
              ip[k-1] = m;
              //  interchange elements in rows k and m. ------------------------------
    double t = a[m-1][k-1];
    if (m != k) {
                        a[m-1][k-1] = a[k-1][k-1];
      a[k-1][k-1] = t;
              }
    if (t == 0.) {
                        ier = k;
      return ier;
              }
              //  store multipliers in a(i,k), i = k+1,...,n. -------------------------
    t = 1./t;
              for ( int i = kp1; i <= n; i++)a[i-1][k-1] = -a[i-1][k-1]*t;
              //  apply multipliers to other columns of a. ----------------------------
    for ( int j = kp1; j <= n; j++) {
                        t = a[m-1][j-1];
      a[m-1][j-1] = a[k-1][j-1];
      a[k-1][j-1] = t;
      if (t != 0.) for ( int i = kp1; i <= n; i++)a[i-1][j-1] = a[i-1][j-1] + a[i-1][k-
1]*t;
              }
       }
       if (a[n-1][n-1] == 0.) ier = n;
  return ier;
}
//---------------------- end of function dec -------------------------


//---------------------------------------------------------------------------
//  solution of linear system a*x = b using output of dec.
//  input..
//     n = order of matrix.
//     ndim = declared first dimension of array  a.
//     a = triangularized matrix obtained from dec.
//     b = right hand side vector.
//     ip = pivot information vector obtained from dec.
//  do not use if dec has set ier .ne. 0.
//  output..
//     b = solution vector, x .
//---------------------------------------------------------------------------
//
//  reference!  a. //. hindmarsh, l. j. sloan, k. w. fong, and
//              g. h. rodrigue,
//              dec/sol!  solution of dense systems of linear
//              algebraic equations,
//              lawrence livermore laboratory report ucid-30137,
//              june 1976.
//---------------------------------------------------------------------------
void sol (int n, double **a, double *b, int *ip) {
       if (n == 1) {
              b[0] = b[0]/a[0][0];
         return;
       }
  int nm1 = n - 1;
       // apply row permutations and multipliers to b. ------------------------
  for ( int k = 1; k <= nm1; k++) {
              //int kp1 = k + 1;
         int m = ip[k-1];
         double t = b[m-1];
         b[m-1] = b[k-1];
         b[k-1] = t;
              for ( int i = k+1; i <= n; i++)b[i-1] = b[i-1] + a[i-1][k-1]*t;
       }
       // back solve. ---------------------------------------------------------
  for ( int kb = 1; kb <= nm1; kb++) {
              int km1 = n - kb;
         int k = km1 + 1;
         b[k-1] = b[k-1]/a[k-1][k-1];
```

```
          double t = -b[k-1];
          for ( int i = 1; i <= km1; i++)b[i-1] = b[i-1] + a[i-1][k-1]*t;
        }
        b[0] = b[0]/a[0][0];
  return;
}
//----------------------- end of function sol -------------------------


// Swap double
void Swap(int *x,int *y) {
        int temp = *x; *x = *y; *y = temp;
}
```

Appendix 4: Test Program: "testArrPointers.cpp"

```
#define EPS 1.0e-6
#include <iostream>
#include <fstream>
#include <math.h>
#include <stdlib.h>
#include <time.h>       // for clock(), CLOCKS_PER_SEC
#include <string.h>
#include <ctype.h>
using namespace std;


void simplx(int iposv[],int i2posv[]);
void subTime(int *&iposv,int *&i2posv);
void subsubinnerTime(int *&iposv,int *&i2posv);
void suballocA(int *&iposv,int *&i2posv);
int *ivector(int nl,int nh);

int main(int argc, char * const argv[])
{
    int *iposv;
    int *i2posv;
    suballocA(iposv,i2posv);
    subTime(iposv,i2posv);
    return 0;
}

void subTime(int *&iposv,int *&i2posv)
{
    subsubinnerTime(iposv,i2posv);
}

void subsubinnerTime(int *&iposv,int *&i2posv)
{
    simplx(iposv,i2posv);
    int i;
    for(i=1;i<10;i++)
    {   printf("Inside subsubinnerTime, i=%d, %d, %d\n",i,iposv[i],i2posv[iposv[i]-1]); }
}

void simplx(int iposv[],int i2posv[])
{
    int i;
    for(i=0;i<10;i++)
    {
        iposv[i] = i;
        i2posv[i]=i*10;
    }
}
void suballocA(int *&iposv,int *&i2posv)
{
   iposv=ivector(10,100);
   i2posv=ivector(10,100);
}

int *ivector(int nl,int nh) {
        int *v;
        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) std::cout<<"allocation failure in ivector()\n";
        return v-nl;
}
```

Appendix 5: HSDMAS "Makefile"

```
SWIG=/Users/nudson/Personal/HSDMAS/cascadeModel/swig/swig-1.3.31//preinst-swig
SWIG_FLAGS=-java -c++

CXX=g++ -O3
CXX_FLAGS=-c -Wall -Wno-long-double -I/System/Library/Frameworks/JavaVM.framework/Headers
REPAST_FLAGS=-DREPAST

JAVAC=javac
JAVAC_FLAGS=-classpath lib/repast.jar:.

LD=g++
LDFLAGS=-Wno-long-double -dynamiclib -framework JavaVM

all: Cascade.class original_cascadeModel original_opaModel

libcascadeModel.jnilib: cascadeModel.o cascadeModel_wrap.o
	$(LD) $^ $(LDFLAGS) -o $@

libopaModel.jnilib: opaModel.o opaModel_wrap.o
	$(LD) $^ $(LDFLAGS) -o $@

cascadeModel.java cascadeModel_wrap.cxx: cascadeModel.i cascadeModel.cpp
	$(SWIG) $(SWIG_FLAGS) cascadeModel.i

opaModel.java opaModel_wrap.cxx: opaModel.i opaModel.cpp
	$(SWIG) $(SWIG_FLAGS) opaModel.i

cascadeModel.o: cascadeModel.cpp
	$(CXX) $(CXX_FLAGS) $(REPAST_FLAGS) cascadeModel.cpp

opaModel.o: opaModel.cpp
	$(CXX) $(CXX_FLAGS) $(REPAST_FLAGS) opaModel.cpp

cascadeModel_wrap.o: cascadeModel_wrap.cxx cascadeModel.java cascadeModelJNI.java
	$(CXX) $(CXX_FLAGS) cascadeModel_wrap.cxx

opaModel_wrap.o: opaModel_wrap.cxx opaModel.java opaModelJNI.java
	$(CXX) $(CXX_FLAGS) opaModel_wrap.cxx

%.class:%.java
	$(JAVAC) $(JAVAC_FLAGS) $<

Cascade.class: Cascade.java cascadeModel.class cascadeModelJNI.class
libcascadeModel.jnilib opaModel.class opaModelJNI.class libopaModel.jnilib
	$(JAVAC) $(JAVAC_FLAGS) Cascade.java

original_cascadeModel: cascadeModel.cpp
	$(CXX) -Wall -o original_cascadeModel cascadeModel.cpp

original_opaModel: opaModel.cpp
	$(CXX) -Wall -o original_opaModel opaModel.cpp


clean:
	rm -f cascadeModel_wrap.cxx cascadeModel_wrap.o libcascadeModel.jnilib
	rm -f opaModel_wrap.cxx opaModel_wrap.o libopaModel.jnilib
	rm -f SWIGTYPE*
	rm -f opaModel.java opaModel.o opaModelJNI.class opaModelJNI.java
	rm -f cascadeModel.java cascadeModel.o cascadeModelJNI.class cascadeModelJNI.java
	rm -f cascadeModel.class Cascade.class original_cascadeModel.o
original_cascadeModel
	rm -f opaModel.class original_opaModel.o original_opaModel
	rm -f AgeDistribution Agent-1 Agent-2 Cascades Failures
	rm -f CascadesAdverse CascadesNonAdverse Time\ Evolution
	rm -f TimeSeries WaitTimes Cascades-q-high Cascades-q-highNonAdverse
```

```
rm -f Averaged Economics1 Economics2 Line_limits iterations_outages
rm -f iterations_powershed
rm -f Marginal_generators Time_var_blackout
rm -f *.class Cascades-q-low
```

Appendix 6: SWIG Wrapper File: "cascadeModel.i"

```
/* File : cascadeModel.i */
%module cascadeModel

%inline %{
extern void initializeSystem();
extern void timeAdvance();
extern int closeFiles();
extern void beforeAgents(int);
extern void afterAgents(int);
extern void updateProfits();

extern int getNextRand();
extern double getRAND_MAX();


double getMaxPValue(int);
void setMaxPValue(int, double);
double getMeanMaxPValue(int);
void setMeanMaxPValue(int, double);
double getMaxTRAValue(int);
void setMaxTRAValue(int, double);
double getMaxdayTValue(int);
void setMaxdayTValue(int, double);
double getMPValue(int);
void setMPValue(int, double);
double getTRAValue(int);
void setTRAValue(int, double);

double getMaxP2Value(int);
void setMaxP2Value(int, double);
double getMeanMaxP2Value(int);
void setMeanMaxP2Value(int, double);
double getMaxmuValue(int);
void setMaxmuValue(int, double);
double getMaxdayValue(int);
void setMaxdayValue(int, double);
double getMP2Value(int);
void setMP2Value(int, double);
double getMUValue(int);
void setMUValue(int, double);

double getMaxP3Value(int);
void setMaxP3Value(int, double);
double getMeanMaxP3Value(int);
void setMeanMaxP3Value(int, double);
double getMaxTRMValue(int);
void setMaxTRMValue(int, double);
double getMaxdayTmValue(int);
void setMaxdayTmValue(int, double);
double getMP3Value(int);
void setMP3Value(int, double);
double getTRMValue(int);
void setTRMValue(int, double);

extern int Jevaluation;
extern int j;
extern int jkk;
extern double meanCost;
extern double TauR;
extern double mu;
extern double TauM;

/*extern int Jmeasure;*/
extern double MaxP;
extern double MaxT;
extern double * MP;
```

```
extern double * TRA;
extern int Ak;
extern int kkj;
extern int kkj3;
extern int ijk;
extern int ijt;
extern int ijt3;
extern double MaxP2;
extern double Maxmu;
extern double pt_MaxP2;
extern double pt_Maxmu;
extern double * MP2;
extern double * MU;
extern int Ak2;
extern int Ak3;
extern int kk2j;
extern double gamma2;
extern int pt_Ak2;
extern int pt_kk2j;
extern double pt_gamma2;
extern double MeanMaxT;
extern double MeanMax;
extern double MeanMaxTm;
extern int je;
extern int jthr;
extern int Iterations;
extern int kc;
extern int Sizeq;
extern int Tauq;
extern double q0;
extern int iq;
extern int Size;
extern int p0;
extern double LoadRange;
extern int Tauf;
extern double XXMMT;
extern double XXmm;
extern double XXMMT3;
%}
```

Appendix 7: SWIG Wrapper File: "opaModel.i"

```
/* File : opaModel.i */
%module opaModel

%inline %{
extern int initializeSystem();
extern void timeLoop();
extern int finalizeSystem();

extern int getNextRand();
extern double getRAND_MAX();

extern int TotalTime;
extern int time_d;
extern double PowerShedTotal;
extern double Dtotal;
extern int overloadTotal;
extern int outages;
%}
```

Appendix 8: HSDMAS REPAST Source Code: "Cascade.java"

```
import uchicago.src.sim.engine.SimpleModel;
import uchicago.src.sim.engine.SimInit;
import uchicago.src.sim.gui.DisplaySurface;
import uchicago.src.sim.gui.Object2DDisplay;
import uchicago.src.sim.space.Object2DTorus;
import uchicago.src.sim.analysis.OpenSequenceGraph;
import uchicago.src.sim.analysis.Sequence;

import java.io.*;
import java.awt.*;
import java.math.*;
import java.lang.Math;
import java.lang.Object;
import java.util.*;
import java.util.Random;

public class Cascade extends SimpleModel{

    public static double g; //used for RANDMAX
    public static Random generator = new Random();
    private DisplaySurface dsurf;
    private int spaceSize=100;
    static double TauR;
    static double mu;
    static double TauM;
    static int j;
    static int Iterations;
    int Tauf;
    int je;
    int jthr;
    int Jevaluation;
    int JevalTemp;
    int Sizeq;
    int Tauq;
    double q0;
    int iq;
    int Size;
    int p0;
    static int jk;
    static int jkk;
    double LoadRange;
    static double RAND_MAX;
    int TotalTime;
    int time_d;
    static double PowerShedTotal;
    static double Dtotal;
    static int overloadTotal;
    static int outages;

static {
    try {
        System.loadLibrary("cascadeModel");
        System.loadLibrary("opaModel");
    }
    catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library failed to load. See chap on Dynamic Linking
Probs in SWIG Java docs for help.\n"+e);
      System.exit(1);
    }
  }
/*** For graph ***/
  private int clusterSz;
  private OpenSequenceGraph plot1;
  private OpenSequenceGraph plot2;
  private OpenSequenceGraph plot3;
```

```
  private OpenSequenceGraph plot4;
  private OpenSequenceGraph plot5;
  private OpenSequenceGraph plot6;
  private OpenSequenceGraph plot7;
  public Cascade() {
      params = new String[] {"Num of Time Steps","Iterations","Failure Time","Replacement
Time","Upgrade Rate mu",
                             "Load Range/Mean Load","tp","System size","tq fixed (0) or
variable (1)","SpaceSize",
                               "Load Shed","Demand","Total Overloads","Outages"};
  }
  public String[] getInitParam() {
      return new String[] {
                       "Iterations",
                       "Failure_Tauf","Replacement_TauR",
                       "UpgradeRate_mu","LoadRange_MeanLoad",
                       "tp","Systemsize","tq","Q_afterBlackout",

"Tauq","Sizeq","SpaceSize","PowerShedTotal","Dtotal","overloadTotal","outages"};

  }
  public void setIterations(int num)
  {      Iterations = num;
  }
  public int getIterations()
  {      return Iterations;
  }
  public void setFailure_Tauf(int num)
  {      Tauf = num;
  }
  public int getFailure_Tauf()
  {      return Tauf;
  }
  public void setReplacement_TauR(double num) { TauR = num; }
  public double getReplacement_TauR() { return TauR; }
  public void setUpgradeRate_mu(double num) {mu = num; }
  public double getUpgradeRate_mu() {return mu; }
  public void setLoadRange_MeanLoad(double num) { LoadRange = num; }
  public double getLoadRange_MeanLoad() {return LoadRange; }
  public void setTp(int num) {p0 = num; }
  public int getTp() { return p0; }
  public void setSystemSize(int num) {Size = num; }
  public int getSystemSize() {return Size; }
  public void setTq(int num) {iq = num; }
  public int getTq() {return iq;}
  public void setQ_afterBlackout(double num) { q0 = num; }
  public double getQ_afterBlackout() { return q0; }
  public void setTauq(int num) { Tauq = num; }
  public int getTauq() {return Tauq; }
  public void setSizeq(int num) {Sizeq = num; }
  public int getSizeq() { return Sizeq; }
  public void setPowerShedTotal(double num) {PowerShedTotal=num;}
  public double getPowerShedTotal() {return PowerShedTotal; }
  public void setDtotal(double num) {Dtotal = num; }
  public double getDtotal() {return Dtotal;}
  public void setOverloadTotal(int num){overloadTotal = num;}
  public int getOverloadTotal(){return overloadTotal;}
  public void setOutages(int num){outages = num;}
  public int getOutages(){return outages;}

  public int getSpaceSize() {return spaceSize;}
  public void setSpaceSize(int spaceSize)
  { this.spaceSize=spaceSize;}

/*** For graph ***/

 //Overrides of SimpleModel
  public void setup() { //called on first start and when setup button is pressed
    super.setup();
```

```
    cascadeModel.initializeSystem();
    opaModel.initializeSystem();
    jk = 0;

/* retrieve initial values from cascadeModel.cpp */
    Iterations = cascadeModel.getIterations();
    j = cascadeModel.getJ();
    Tauf = cascadeModel.getTauf();
    TauR = cascadeModel.getTauR();
    TauM = cascadeModel.getTauM();
    mu = cascadeModel.getMu();
    LoadRange = cascadeModel.getLoadRange();
    p0 = cascadeModel.getP0();
    Size = cascadeModel.getSize();
    iq = cascadeModel.getIq();
    q0 = cascadeModel.getQ0();
    Tauq = cascadeModel.getTauq();
    Sizeq = cascadeModel.getSizeq();

/* retrieve initial values from opaModel.cpp */
    TotalTime = opaModel.getTotalTime();
    time_d = opaModel.getTime_d();
    PowerShedTotal = opaModel.getPowerShedTotal();
    Dtotal = opaModel.getDtotal();
    overloadTotal = opaModel.getOverloadTotal();
    outages = opaModel.getOutages();

/* create graphs for visual representation of changing values */
    if (plot1 != null) plot1.dispose();
    plot1 = new OpenSequenceGraph("TauR", this);
    plot1.setAxisTitles("time", "Calculated Value");
    plot1.addSequence("TauR", new Sequence() {
      public double getSValue() { return TauR; }
    });

    if (plot2 != null) plot2.dispose();
    plot2 = new OpenSequenceGraph("mu", this);
    plot2.setAxisTitles("time", "Calculated Value");
    plot2.addSequence("Mu", new Sequence() {
      public double getSValue() { return (mu-1); }
    });

    if (plot3 != null) plot3.dispose();
    plot3 = new OpenSequenceGraph("TauM", this);
    plot3.setAxisTitles("time", "Calculated Value");
    plot3.addSequence("TauM", new Sequence() {
      public double getSValue() { return TauM; }
    });

    if (plot4 != null) plot4.dispose();
    plot4 = new OpenSequenceGraph("PowerShedTotal", this);
    plot4.setAxisTitles("time", "Calculated Value");
    plot4.addSequence("PowerShedTotal", new Sequence() {
      public double getSValue() { return PowerShedTotal; }
    });

    if (plot5 != null) plot5.dispose();
    plot5 = new OpenSequenceGraph("Dtotal", this);
    plot5.setAxisTitles("time", "Calculated Value");
    plot5.addSequence("Dtotal", new Sequence() {
      public double getSValue() { return Dtotal; }
    });

    if (plot6 != null) plot6.dispose();
    plot6 = new OpenSequenceGraph("overloadTotal", this);
    plot6.setAxisTitles("time", "Calculated Value");
    plot6.addSequence("overloadTotal", new Sequence() {
      public double getSValue() { return overloadTotal; }
    });
```

```
    if (plot7 != null) plot7.dispose();
    plot7 = new OpenSequenceGraph("outages", this);
    plot7.setAxisTitles("time", "Calculated Value");
    plot7.addSequence("outages", new Sequence() {
      public double getSValue() { return outages; }
    });

    plot1.setXRange(0, 50);
    plot1.setYRange(0, 2);
    plot1.display(); //pop-up the plot window
    plot2.setXRange(0, 50);
    plot2.setYRange(0, 0.3);
    plot2.display(); //pop-up the plot window
    plot3.setXRange(0, 50);
    plot3.setYRange(0, 2);
    plot3.display(); //pop-up the plot window
    plot4.setXRange(0,50);
    plot4.setYRange(-5,25);
    plot4.display(); //pop-up plot window
    plot5.setXRange(0,50);
    plot5.setYRange(500,2500);
    plot5.display(); //pop-up plot window
    plot6.setXRange(0,50);
    plot6.setYRange(0,50);
    plot6.display(); //pop-up plot window
    plot7.setXRange(0,50);
    plot7.setYRange(0,50);
    plot7.display(); //pop-up plot window
}

public void buildModel() { //called when simulation buttons run, step, or initialize
buttons presssed
/* create independent agents here and add to array called agentList
   Agent A1 = new Agent(...);
   Agent A2 = new Agent(...);
   Agent A3 = new Agent(...);
   agentList.add(A1);
   agentList.add(A2);
   agentList.add(A3);
*/
}

public static void main(String[] args)
{

    RAND_MAX = cascadeModel.getRAND_MAX();
    SimInit init = new SimInit();
    init.loadModel(new Cascade(), null, false);
}

int count = 0;
int subcount = 0;
int subcountopa = 0;
public void step()
{
    /* set a seed for random number generator */
    //long sd = 590044025;
    //generator.setSeed(sd);

if (count <= 200)
{
    subcount = 0;
    subcountopa = 0;
    plot1.step(); //plot current value of TauR
    plot2.step(); //plot current value of mu
    plot3.step(); //plot current value of TauM
    plot4.step(); //plot current value of PowerShedTotal or "Load Shed"
    plot5.step(); //plot current value of Dtotal or "Demand"
```

```
    plot6.step(); //plot current value of overloadTotal or "Total overloads"
    plot7.step(); //plot current value of outages
    PowerShedTotal = opaModel.getPowerShedTotal();
    Dtotal = opaModel.getDtotal();
    overloadTotal = opaModel.getOverloadTotal();
    outages = opaModel.getOutages();

    while(subcountopa <=500) // iterate through opa 500 times per 1 repast step
    { if(time_d <= TotalTime)
      {
         opaModel.timeLoop();
         time_d++;
         subcountopa++;
      }
      else { break; }
    }

   System.out.println("finished 500 of opa, count="+count+" time_d="+time_d+"
TotalTime="+TotalTime+"\n");
    while(subcount <= 5000)
// iterate through cascade 5000 times per 1 repast step
    {                       // will eventually interweave opa and cascade loops
      if ( j < Iterations)
      {
         cascadeModel.timeAdvance();
         jthr = cascadeModel.getJthr();
         if(j > jthr)
         {
              je = cascadeModel.getJe();
              je++;
              cascadeModel.setJe(je);
              Jevaluation = cascadeModel.getJevaluation();
              JevalTemp = Jevaluation -1;
              if ( je > JevalTemp)
              {
                    cascadeModel.beforeAgents(jk);
                    jkk = cascadeModel.getJkk();
                    //TauR = 15;
                    TauR = callAgent1();
                    cascadeModel.setTauR(TauR);
                    //mu = 1.009;
                    mu = callAgent2();
                    cascadeModel.setMu(mu);
                    TauM = callAgent3();
                    cascadeModel.setTauM(TauM);
                    cascadeModel.afterAgents(jk);
                    double meanCost = 0.0;
                    cascadeModel.setMeanCost(meanCost);
                    jk++;
                    jkk++;
                    cascadeModel.setJkk(jkk);
              }
              else
              {
                     cascadeModel.updateProfits();
              }
          }
         int tempj = cascadeModel.getJ();
         tempj++;
         int tempkc = cascadeModel.getKc();
         tempkc++;
         cascadeModel.setJ(tempj);
         cascadeModel.setKc(tempkc);
         j = tempj;

         JevalTemp = cascadeModel.getJevaluation();
    } //end if
    subcount++;
    } //end while(subcount)
```

```
count++;
}
else{
    System.out.println("Total iterations is "+Iterations+"\n");
    cascadeModel.closeFiles();
    opaModel.finalizeSystem();
    getController().pauseSim();}
}
public static double callAgent1()
{
      int     kkkj    = cascadeModel.getKkj();;
      int Nexplor     = 22;
      int Ntrans      = Nexplor*cascadeModel.getJevaluation();
      int kijt        = cascadeModel.getIjt();
      double MeanMax1         = cascadeModel.getMeanMaxT();
      if (jk <= 1)
      {
          kijt    = 1;
          MeanMax1        = 0.;
      }
      double MP_jk = cascadeModel.getMPValue(jk);
      int j = cascadeModel.getJ();
      if (MP_jk > MeanMax1 && ((j < Ntrans + 1) || (j > Ntrans)))
      {
          kijt  = kijt + 1;
          cascadeModel.setMaxPValue(kijt,MP_jk);
          cascadeModel.setMaxTRAValue(kijt,cascadeModel.getTRAValue(jk));
          cascadeModel.setMaxdayTValue(kijt,jk);
          cascadeModel.setMeanMaxPValue(kijt-1,MeanMax1);
          MeanMax1   = 0.0;
      }

      double XX  = jk - cascadeModel.getMaxdayTValue(kijt);
      MeanMax1   = (MP_jk+XX*MeanMax1)/(XX+1.0);

      if (MeanMax1 < 0.98*cascadeModel.getMeanMaxPValue(kijt-1))
      {
          cascadeModel.setMeanMaxPValue(kijt,MeanMax1);
          int jmax                = 1;
          XX              = 0.;
          double XY               = 0.;
          for (int i = 2 ; i<kijt + 1; i++)
          {
               if (cascadeModel.getMeanMaxPValue(i)>cascadeModel.getMeanMaxPValue(i-1) )
                  jmax    = i;
               XX      = XX +
                         cascadeModel.getMeanMaxPValue(i)*cascadeModel.getMaxTRAValue(i);
               XY      = XY + cascadeModel.getMeanMaxPValue(i);
          }
         if (jmax == 1) jmax = kijt;
         kijt                    = kijt + 1;
         //if ( (cascadeModel.getNextRand()/RAND_MAX) > 0.25)
         if ( generator.nextDouble() > 0.25)
         {
              cascadeModel.setMaxPValue(kijt,cascadeModel.getMeanMaxPValue(jmax));
              cascadeModel.setMaxTRAValue(kijt,cascadeModel.getMaxTRAValue(jmax));
              cascadeModel.setMaxdayTValue(kijt,jk);
         }
         else
         {
              cascadeModel.setMaxPValue(kijt, XY/(kijt-2));
              cascadeModel.setMaxTRAValue(kijt,2.);
              if (XY > 0) cascadeModel.setMaxTRAValue(kijt, XX/XY);
              cascadeModel.setMaxdayTValue(kijt,jk);
         }
         MeanMax1 = cascadeModel.getMeanMaxPValue(jmax);
     }
     double MMT      = cascadeModel.getMaxTRAValue(kijt);
     if (MMT < 1) MMT        = 2.;
```

```
     if (jk < 31 || jkk < 3) TauR    = TauR - 6;
     if (jk > 30 && jkk > 2)
     {
         double A        = 1.0;
         double B        = 0.0;
         double TRAjk_1 = cascadeModel.getTRAValue(jk-1);
         double TRAjk_2 = cascadeModel.getTRAValue(jk-2);
         double TRAjk_3 = cascadeModel.getTRAValue(jk-3);
         double MPjk_1 = cascadeModel.getMPValue(jk-1);
         double MPjk_2 = cascadeModel.getMPValue(jk-2);
         double MPjk_3 = cascadeModel.getMPValue(jk-3);
         if( Math.abs(TRAjk_1-TRAjk_2) > 0.0001 && Math.abs(TRAjk_1-TRAjk_3) > 0.0001 &&
                Math.abs(TRAjk_2-TRAjk_3)>0.0001)
         {
              A       = ((MPjk_1-MPjk_2)/(TRAjk_1-TRAjk_2)-(MPjk_1-MPjk_3)/
                                          (TRAjk_1-TRAjk_3))/(TRAjk_2-TRAjk_3);
              B       =(MPjk_1-MPjk_2)/(TRAjk_1-TRAjk_2)-A*(TRAjk_1+TRAjk_2);
          }

         //std::cout << A << "\t" << B <<   "\n";

        if (A < 0)
        {
             TauR            = -B/(2.*A);
             cascadeModel.setAk(1);
        }

        else
       {
            cascadeModel.setAk(2);
            TauR            = MMT;

            //if ( cascadeModel.getNextRand()/RAND_MAX > 0.90)
            if ( generator.nextDouble() > 0.90)
            {
                //TauR    = MMT + 10.*(cascadeModel.getNextRand()/RAND_MAX-0.5);
                TauR    = MMT + 10.*(generator.nextDouble()-0.5);
                kkkj++;
                cascadeModel.setAk(3);
            }
        }
        if (TauR < 0) TauR      = 2.;
        //if (TauR > 10.*MMT) TauR        = MMT +
                                      10.*((cascadeModel.getNextRand()/RAND_MAX)-0.5);
        if (TauR > 10.*MMT) TauR        = MMT + 10.*(generator.nextDouble()-0.5);
    }


    cascadeModel.setKkj(kkkj);
    if (TauR > 500) TauR    = 500.;
    if (TauR < 1) TauR      = 2.;
    cascadeModel.setXXMMT(MMT);
    cascadeModel.setMeanMaxT(MeanMax1);
    cascadeModel.setIjt(kijt);

    return TauR;
}

public static double callAgent2()
{
   int Nexplor     = 22;
   int Ntrans      = Nexplor*cascadeModel.getJevaluation();
   int kijk        = cascadeModel.getIjk();
   double Xmu      = cascadeModel.getMu() - 1.0;
   double Xg       = cascadeModel.getGamma2();
   double MeanMax1 = cascadeModel.getMeanMax();
   if (jk <= 1)
   {
       kijk    = 1;
```

```
    MeanMax1        = 0.;
}
double MP2_jk = cascadeModel.getMP2Value(jk);
int j = cascadeModel.getJ();
if ((MP2_jk > MeanMax1 && j < Ntrans + 1)||(MP2_jk > 1.05*MeanMax1 && j > Ntrans))
{
       kijk                          = kijk + 1;
       cascadeModel.setMaxP2Value(kijk,MP2_jk);
       cascadeModel.setMaxmuValue(kijk,cascadeModel.getMUValue(jk));
       cascadeModel.setMaxdayValue(kijk,jk);
       cascadeModel.setMeanMaxP2Value(kijk-1,MeanMax1);
       MeanMax1                      = 0.0;
}

double XX       = jk - cascadeModel.getMaxdayValue(kijk);
MeanMax1        = (MP2_jk+XX*MeanMax1)/(XX+1.0);

if (MeanMax1 < 0.95*cascadeModel.getMeanMaxP2Value(kijk-1))
{
     cascadeModel.setMeanMaxP2Value(kijk,MeanMax1);
     int jmax               = 1;
     XX              = 0.;
     double XY              = 0.;
     for (int i = 2 ; i<kijk + 1; i++)
     {
        if (cascadeModel.getMeanMaxP2Value(i)>cascadeModel.getMeanMaxP2Value(i-1) )
            jmax  = i;
        XX      = XX +
             cascadeModel.getMeanMaxP2Value(i)*cascadeModel.getMaxmuValue(i);
        XY      = XY + cascadeModel.getMeanMaxP2Value(i);
     }
     if (jmax == 1) jmax = kijk;
     kijk                   = kijk + 1;
     //if ( (cascadeModel.getNextRand()/RAND_MAX) > 0.25)
     if ( generator.nextDouble() > 0.25)
     {
         cascadeModel.setMaxP2Value(kijk,cascadeModel.getMeanMaxP2Value(jmax));
         cascadeModel.setMaxmuValue(kijk,cascadeModel.getMaxmuValue(jmax));
         cascadeModel.setMaxdayValue(kijk,jk);
     }
     else
     {
          cascadeModel.setMaxP2Value(kijk,XY/(kijk-2));
          cascadeModel.setMaxmuValue(kijk,cascadeModel.getMaxmuValue(jmax));
          if (XY > 0) cascadeModel.setMaxmuValue(kijk,XX/XY);
          cascadeModel.setMaxdayValue(kijk,jk);
     }
     MeanMax1 = cascadeModel.getMeanMaxP2Value(jmax);

}

double MMU      = cascadeModel.getMaxmuValue(kijk);
if (MMU < Xg-1) MMU     = 3.*(Xg-1);
if (jk < Nexplor + 1 || jkk < 3) Xmu    = Xmu*1.4;
if (jk > Nexplor && jkk > 2)
{
     double A        = 1.0;
     double B        = 0.0;
     double MUjk_1 = cascadeModel.getMUValue(jk-1);
     double MUjk_2 = cascadeModel.getMUValue(jk-2);
     double MUjk_3 = cascadeModel.getMUValue(jk-3);
     double MP2jk_1 = cascadeModel.getMP2Value(jk-1);
     double MP2jk_2 = cascadeModel.getMP2Value(jk-2);
     double MP2jk_3 = cascadeModel.getMP2Value(jk-3);

     if( Math.abs(MUjk_1-MUjk_2) > 0.000001 && Math.abs(MUjk_1-MUjk_3) > 0.000001 &&
            Math.abs(MUjk_3-MUjk_2)>0.000001)
     {
```

```
            A       = ((MP2jk_1-MP2jk_2)/(MUjk_1-MUjk_2)-(MP2jk_1-MP2jk_3)/
                      (MUjk_1-MUjk_3))/(MUjk_2-MUjk_3);
            B       =(MP2jk_1-MP2jk_2)/(MUjk_1-MUjk_2)-A*(MUjk_1+MUjk_2);
         }
        if (A < 0)
        {
            Xmu             = -B/(2.*A);
            cascadeModel.setAk2(1);
         }

         else
         {
             cascadeModel.setAk2(2);
             Xmu                    = MMU;
             //if ( cascadeModel.getNextRand()/RAND_MAX > 0.95)
             if ( generator.nextDouble() > 0.95)
             {
                // Xmu     = MMU*(1.0 + 2.*(cascadeModel.getNextRand()/RAND_MAX-0.5));
                 Xmu     = MMU*(1.0 + 2.*(generator.nextDouble()-0.5));
                 cascadeModel.setAk2(3);
             }
         }

     }
     if (Xmu <  Xg-1) Xmu    = 3.*(Xg-1);
     if (Xmu > 1.) Xmu       = 1.0;

     cascadeModel.setXXmm(MMU);
     cascadeModel.setIjk(kijk);
     cascadeModel.setMeanMax(MeanMax1);
                       //std::cout << MeanMax1 << "\t" << kijk <<   "\n";

     return 1.+Xmu;
}


public static double callAgent3()
{
   int kkkj = 1; /*cascadeModel.getKkj3();*/
   int Nexplor     = 22;
   int Ntrans      = Nexplor*cascadeModel.getJevaluation();
   int kijt        = cascadeModel.getIjt3();
   double MeanMax1  = cascadeModel.getMeanMaxTm();
   if (jk <= 1)
   {
       kijt    = 1;
       MeanMax1        = 0.;
   }
   double MP3_jk = cascadeModel.getMP3Value(jk);
   if (MP3_jk > MeanMax1 && ((j < Ntrans + 1)||(j>Ntrans)))
   {
       kijt                            = kijt + 1;
       cascadeModel.setMaxP3Value(kijt,MP3_jk);
       cascadeModel.setMaxTRMValue(kijt,cascadeModel.getTRMValue(jk));
       cascadeModel.setMaxdayTmValue(kijt,jk);
       cascadeModel.setMeanMaxP3Value(kijt-1,MeanMax1);
       MeanMax1  = 0.0;
   }
   double XX       = jk - cascadeModel.getMaxdayTmValue(kijt);
   MeanMax1        = (cascadeModel.getMP3Value(jk)+XX*MeanMax1)/(XX+1.0);

   if (MeanMax1 < 0.98*cascadeModel.getMeanMaxP3Value(kijt-1))
   {
        cascadeModel.setMeanMaxP3Value(kijt,MeanMax1);
        int jmax                = 1;
        XX              = 0.;
        double XY               = 0.;
        for (int i = 2 ; i<kijt + 1; i++)
        {
```

```
        if (cascadeModel.getMeanMaxP3Value(i)>cascadeModel.getMeanMaxP3Value(i-1))
            jmax  = i;
        XX      = XX +
            cascadeModel.getMeanMaxP3Value(i)*cascadeModel.getMaxTRMValue(i);
        XY      = XY + cascadeModel.getMeanMaxP3Value(i);
    }
    if (jmax == 1) jmax = kijt;
     kijt                    = kijt + 1;
    //if ( cascadeModel.getNextRand()/RAND_MAX > 0.25)
    if ( generator.nextDouble() > 0.25)
    {
          cascadeModel.setMaxP3Value(kijt,cascadeModel.getMeanMaxP3Value(jmax));
          cascadeModel.setMaxTRMValue(kijt,cascadeModel.getMaxTRMValue(jmax));
          cascadeModel.setMaxdayTmValue(kijt,jk);
    }
    else
    {
         cascadeModel.setMaxP3Value(kijt,XY/(kijt-2));
         cascadeModel.setMaxTRMValue(kijt,2.);
         if (XY > 0) cascadeModel.setMaxTRMValue(kijt,XX/XY);
         cascadeModel.setMaxdayTmValue(kijt,jk);
    }
    MeanMax1 = cascadeModel.getMeanMaxP3Value(jmax);

}

double MMT      = cascadeModel.getMaxTRMValue(kijt);
if (MMT < 1) MMT        = 2.;
if (jk < 31 || jkk < 3) TauM    = TauM - 4;
if (jk > 30 && jkk > 2)
{
   double A        = 1.0;
   double B        = 0.0;
   double TRMjk_1 = cascadeModel.getTRMValue(jk-1);
   double TRMjk_2 = cascadeModel.getTRMValue(jk-2);
   double TRMjk_3 = cascadeModel.getTRMValue(jk-3);
   double MP3jk_1 = cascadeModel.getMP3Value(jk-1);
   double MP3jk_2 = cascadeModel.getMP3Value(jk-2);
   double MP3jk_3 = cascadeModel.getMP3Value(jk-3);
   if(Math.abs(TRMjk_1-TRMjk_2) > 0.0001 && Math.abs(TRMjk_1-TRMjk_3) > 0.0001 &&
            Math.abs(TRMjk_2-TRMjk_3)>0.0001)
   {

       A = ((MP3jk_1-MP3jk_2)/(TRMjk_1-TRMjk_2)-(MP3jk_1-MP3jk_3)/
                   (TRMjk_1-TRMjk_3))/(TRMjk_2-TRMjk_3);
       B =(MP3jk_1-MP3jk_2)/(TRMjk_1-TRMjk_2)-A*(TRMjk_1+TRMjk_2);
   }

                            //std::cout << A << "\t" << B <<   "\n";

  if (A < 0)
  {
       TauM            = -B/(2.*A);
       cascadeModel.setAk3(1);
  }
  else
  {
       cascadeModel.setAk3(2);
       TauM            = MMT;


       //if ( cascadeModel.getNextRand()/RAND_MAX > 0.90)
       if ( generator.nextDouble() > 0.90)
       {
          //TauM    = MMT + 10.*(cascadeModel.getNextRand()/RAND_MAX-0.5);
          TauM    = MMT + 10.*(generator.nextDouble()-0.5);
          kkkj++;
          cascadeModel.setAk3(3);
       }
```

```
      }
      if (TauM < 0) TauM      = 2.;
      //if (TauM > 10.*MMT) TauM = MMT + 10.*(cascadeModel.getNextRand()/RAND_MAX-0.5);
      if (TauM > 10.*MMT) TauM = MMT + 10.*(generator.nextDouble()-0.5);
   }
   cascadeModel.setKkj3(kkkj);
   if (TauM > 500) TauM    = 500.;
   if (TauM < 1) TauM      = 1.;
   cascadeModel.setXXMMT3(MMT);
   cascadeModel.setMeanMaxTm(MeanMax1);
   cascadeModel.setIjt3(kijt);

               return TauM;
}
}
```